VŠB-Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra 460 - Katedra informatiky

SNMP daemon pro DVB kartu v OS Linux

SNMP Daemon for DVB Cards in OS Linux

Akademický rok 2014/2015 **Lukáš Zajac**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lukáš Zajac**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: SNMP daemon pro DVB kartu v OS Linux
SNMP Daemon for DVB Cards in OS Linux

Zásady pro vypracování:

Pro informaci o aktuálním stavu zařízení v počítačové síti je zvykem využívat SNMP protokol. Pokud je v počítačové síti používán streamovací server který používá DVB karty je vhodné sledovat i stav těchto karet. Cílem práce je vytvořit daemona pro OS Linux, který bude sledovat stav DVB karet a bude poskytovat informace o stavu DVB karet prostřednictvím SNMP protokolu.

1. Stručně popište protokol SNMP.
2. Stručně popište DVB API pro OS Linux.
3. Na základě MIB tabulky pro DVB zařízení určete které informace o DVB kartě je reálné získat pomocí DVB API.
4. Implementujte program, který bude zpracovávat informace získané z DVB API a bude data prezentovat v podobě SNMP serveru. Zvažte spolupráci se standardním Linuxovým daemonem snmpd.
5. Pro instalaci vytvořte instalační balíček.
6. Vytvořte plugin pro systém CACTI, který bude prezentovat všechna data získávána z DVB karty.

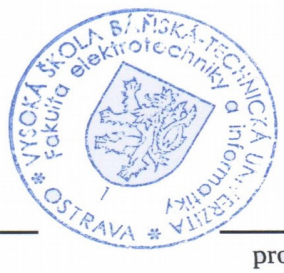
Seznam doporučené odborné literatury:

[1] MAURO, Douglas R a Kevin J SCHMIDT. Essential SNMP. 2nd ed. Beijing: O´Reilly, 2005, 442 s. ISBN 05-960-0840-6.
[2] SKOČOVSKÝ, Luděk a Scott JERNIGAN. Linux: dokumentační projekt. 4., aktualiz. vyd. Překlad Ivo Fořt, David Krásenský. Brno: Computer Press, 2007, 1334 s. ISBN 978-80-251-1525-1.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Seidl, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2015



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

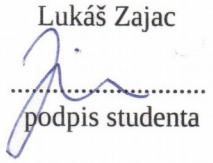
prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 15.4.2015

Lukáš Zajac



.....

podpis studenta

Poděkování

Tímto děkuji svému vedoucímu bakalářské práce, Ing. Davidu Seidlovi, Ph.D., za rady, připomínky a náměty. Rovněž mu děkuji za konzultace mimo období výuky.

Abstrakt a klíčová slova

Abstrakt:

Práce si klade za cíl vytvořit nástroj pro monitorování přijímaného signálu vysílacího serveru, který šíří vysílání přijímané prostřednictvím DVB standartu. Za účelem vzdáleného monitoringu se využije známého síťového protokolu SNMP a finální aplikace rozšíří existující linuxový SNMP démon *snmpd*. DVB konsorcium definovalo několik MIB souborů zabývajících se informacemi o parametrech DVB vysílání, ale ty jsou určeny pro speciální a cenově nákladné měřící karty. Práce se zaobírá výběrem maxima vhodných informací z MIB souborů, které lze implementovat bez využití měřících karet, pouze pomocí DVB API pro OS Linux. Pro instalaci monitorovací aplikace je za cílem co největší uživatelské přívětivosti zvolen standardizovaný balíčkovací systém, v tomto případě balíček pro Debian. Prezentaci dynamických informací na monitorovacích stanicích zajistí rozšiřující modul pro monitorovací nástroj Cacti.

Klíčová slova:

SNMP, MIB, DVB, Linux, snmpd, C++, C, daemon, Net-SNMP, V4L

Abstract:

The work aims to create an instrument for monitoring the received signal of stream server, which distributes broadcast received through the DVB standard. For the puropose of remote monitoring is used familiar network protocol SNMP and final applications extend existing Linux SNMP daemon *snmpd*. DVB consortium has defined several MIB files dealing with information about the parameters of DVB broadcasts, but those are designed for special and very expensive measuring cards. The work deals with the selection of maximum appropriate information from MIB files that can be implemented without the use of measuring cards, using only DVB API on OS Linux. To install a monitoring application is choosed the most user-friendly way of standardized packaging system, in this case the package for Debian. Presentation of dynamic information on monitoring stations provide a plugin for Cacti monitoring tool.

Key words:

SNMP, MIB, DVB, Linux, snmpd, C++, C, daemon, Net-SNMP, V4L

Seznam zkratek a symbolů

ATM	Asynchronous Transfer Mode, síťový standard pro přenos dat.
ASN.1	Abstract Syntax Notation One, popis datových struktur používaných pro reprezentaci, kódování, přenos, ukládání a dekódování dat v telekomunikacích, počítačových sítích a Informatice.
BER	Basic Encoding Rules, formát kódování definován v ASN.1.
BER	Bit Error Rate, počet chybných bitů z celkového počtu přijatých.
BGP	Border Gateway Protocol, síťový routovací protokol.
CAT	Conditional Access Table, jedna z PSI/SI tabulek.
CRC	Cyclic Redundancy Check, hashovací funkce pro kontrolu správnosti přenášených dat.
DVB	Digital Video Broadcasting, digitálně šířené vysílání.
DNS	Domain Name System, hierarchický systém doménových jmen.
ECM	Entitlement Control Message, zpráva související se systémem podmíněného přístupu v DVB.
EMM	Entitlement Management Message, zpráva související se systémem podmíněného přístupu v DVB.
EIT	Event Information Table, jedna z PSI/SI tabulek.
XML	Extensible Markup Language, rozšiřitelný značkovací jazyk, obecný značkovací jazyk pro výměnu dat mezi aplikacemi.
IETF	Internet Engineering Task Force, organizace vyvíjející a starající se o internetové protokoly.
IPTV	Internet Protocol television, televize přes internetový protokol.
MIB	Management Information Base, popisuje sadu objektů, které jsou předmětem správy v SNMP.
MPEG	Moving Picture Experts Group, skupina zaobírající se standarty zpracování audiovizuálního obsahu.
NIT	Network Information Table, jedna z PSI/SI tabulek.
NMA	Network Management Applications, aplikace pro správu sítí.
NMS	Network Management Station, stanice provozující aplikace pro monitorování (NMA).
OID	Object Identifier, identifikátor uzlů v MIB stromu.
OS	Operating System, operační sytém.
PID	Packet Identifier, číslo sloužící k identifikaci paketu.
PAT	Program Association Table, jedna z PSI/SI tabulek
PMT	Program Map Table, jedna z PSI/SI tabulek.
PSI/SI	Program Specific Infomartion/Specific Information, informační tabulky obsažené v transportním proudu (TS).

RFC	Request For Comments, sada dokumentů popisující internetové protokoly, jsou brány pouze jako doporučení.
SDT	Service Description Table, jedna z PSI/SI tabulek.
SNR	Signal to Noise Ratio, poměr síly signálu k síle šumu.
SNMP	Simple Network Management Protocol, protokol používaný nejčastěji pro správu sítě
SMI	Structure of Management Information, definuje jakým způsobem popisovat objekty v SNMP.
TDT	Time and Date Table, jedna z PSI/SI tabulek.
TCP	Transmission Control Protocol, internetový protokol transportní vrstvy.
TSDT	Transport Stream DescriptorTable, jedna z PSI/SI tabulek.
TS	Transport Stream, přenosový proud.
UDP	User Datagram Protocol, internetový protokol transportní vrstvy.

Obsah

1 Úvod

V době rapidního rozšiřování dostupnosti vysokorychlostního připojení k internetu, vznikají nové možnosti šíření multimediálního obsahu jako je IPTV. Poskytovatel na svém serveru (DVB vysílací server) přijímá televizní signál nejčastěji prostřednictvím DVB-T nebo DVB-S vysílání a šíří ho dále přes IPTV. S tímto vzniká potřeba vzdáleně monitorovat DVB přijímače na serveru, pro zajištění kvalitních a dostupných služeb.

Cílem mé práce bylo vytvořit monitorovací aplikaci, která získává informace o stavu DVB karet a tyto informace dále nabízela prostřednictvím síťového SNMP protokolu monitorujícím stanicím (NMS). SNMP server poskytuje informace definované v MIB souborech. Pro účely monitorování parametrů DVB bylo definováno několik MIB souborů, které jsou určeny pro speciální a cenově náročné měřící karty. Já se pokusím implementovat co nejvíce informací jen s použitím DVB API pro OS Linux.

Z vytvořených aplikací jsem sestavil standardizovaný instalační balíček (debian), díky kterému i nepříliš pokročilý uživatel jednoduše zprovozní aplikaci na svém serveru. Dynamické parametry se poté budou prezentovat v nástroji Cacti[6].

2 Protokol SNMP

Simple Network Management Protocol, zkráceně SNMP, je standardizovaný internetový protokol pro údržbu zařízení v IP sítích. Podpora SNMP protokolu je integrována do zařízení jako routery, switche, servery, uživatelské stanice, tiskárny, záložní zdroje energie a mnohá další, u kterých to má smysl. SNMP protokol umožňuje nejen vzdáleně monitorovat stav zařízení, ale podporuje i možnost vzdáleného ovládání, což ukazuje jeho obousměrnost.

2.1 Obecně o SNMP protokolu

2.1.1 Vývoj SNMP protokolu

O standard SNMP protokolu se stará organizace Internet Engineering Task Force (IETF), která je jako všechny ostatní internetové protokoly definovala v Ready for Comments (RFC) dokumentech.

SNMP standart vyšel prozatím ve třech verzích:

- SNMP verze 1 (SNMPv1)
    - nízké zabezpečení, přenos hesla v čistém textu
    - definován v RFC 1157
- SNMP verze 2 (SNMPv2)
    - definován v RFC 3416, RFC 3417, a RFC 3418
- SNMP verze 3 (SNMPv3)
    - oficiálně vydán v roce 2002
    - podpora zabezpečené autentizace
    - soukromá komunikace mezi spravovanými zařízeními
    - definován v RFC 3410, RFC 3411, RFC 3412, RFC 3413, RFC 3414, RFC 3415, RFC 3416, RFC 3417, RFC 3418, a RFC 2576

2.1.2 Manažeři a agenti

V síti se v souvislosti se SNMP protokolem vyskytují zařízení plnící dvě rozličné role. Jsou to takzvaní agenti a manažeři.

Manažerem je míněn server se speciálním softwarem, jehož úkolem je zpracovávat úkoly pro síť. Manažeři se v terminologii SNMP nazývají Network Management Stations (NMSs). NMS je

odpovědný za zasílání dotazů a příjem trapů od agentů (router, switch, Unixový server...). Trap je způsob, jakým může agent oznámit NMS nějakou událost asynchronně, čili se na ni NMS nemusí ptát. NMS může dále reagovat na informace získané od agentů a provádět na ně odpovídající reakce.

Agent je zařízení v síti vybaveno potřebným softwarem. Nejčastěji to bývá nezávislý program zvaný démon, nebo je integrován přímo do firmwaru zařízení. V současné době má mnoho běžných síťových zařízení integrováno nějaký druh SNMP agenta.

Vztah a komunikace mezi NMS a agentem je názorně vysvětlena na obrázku 1.

Obrázek 1: Vztah mezi agentem a NMS [1]

Trap a požadavek můžou nastat v jakoukoliv dobu, nebo dokonce oba ve stejný čas.

2.1.3 Struktura správy informací a MIB

Struktura správy informací (SMI - Structure of Management Information), poskytuje směr, jakým způsobem spravovat objekt a jeho chování. Agent má seznam objektů které mapuje. Objektem může být například port routeru a jeho chováním je jeho stav (aktivní, neaktivní).

Management Information Base (MIB) je databáze spravovaných objektů, které agent mapuje. Všechny informace, které může NMS získat jsou uvedené v MIB. SMI je způsob jakým se mají objekty pospat a MIB jsou konkrétní popsané objekty za použití syntaxe SMI.

Agent může implementovat mnoho MIB, ale všichni agenti implementují základní MIB zvané MIB-II, jehož objekty mají informace o daném agentovi (stav síťového rozhraní, umístění...).

Existuje mnoho MIB pro různá zařízení a servery poskytující služby, jako jsou například:

- ATM MIB (RFC 2515)
- Frame Relay DTE Interface Type MIB (RFC 2115)

- BGP Version 4 MIB (RFC 1657)
- RDBMS MIB (RFC 1697)
- RADIUS Authentication Server MIB (RFC 2619)
- Mail Monitoring MIB (RFC 2789)
- DNS Server MIB (RFC 1611)

Každý výrobce si může pro své zařízení definovat své vlastní proprietární MIB a umožnit tak přístup k jeho speciálním objektům.

2.1.4 Koncept síťového managementu

SNMP je vytvořen pro podporu správy sítí. Správa sítí je disciplína, která za použití nejrůznějších technik, postupů a nástrojů umožňuje lidem udržovat rozličná zařízení, systémy a sítě. Model správy sítě podle ISO standardu vypadá následovně:

- Fault Management (Správa chyb)

  - detekování, logování, a informování uživatelů o problému

- Configuration Management (Správa konfigurace)

  - správa a monitorování systémové konfigurace (volná RAM, verze firmwaru...)

- Accounting Management (Správa uživatelského přístupu)

  - zajištění spravedlivého přístupu uživatelů k systémovým a síťovým prostředkům

- Performance Management (Správa výkonu)

  - měření a reportování různých výkonových charakteristik sítě nebo sytému

- Security Management (Správa zabezpečení)

  - řízení přístupu uživatelů ke zdrojům a snaha zamezit útokům o proniknutí do sítě a znepřístupnění služby

K zajištění plnění všech funkcí vrstev modelu se dnes používá možností a funkcionalit protokolu SNMP, který byl navrhnut právě za tímto účelem.

2.2 SNMPv1 a SNMPv2

SNMPv1 a SNMPv2 jsou si velmi podobné svými vlastnostmi. SNMPv3 přidává lepší možnosti zabezpečení a mnoho jiných vylepšení, tyto vlastnosti ale již neovlivňují mou další práci a tedy se

nebudu verzi 3 dále teoreticky věnovat.

2.2.1 Protokol transportní vrstvy

SNMP využívá jako transportní protokol UDP, jelikož UDP nezasílá potvrzení o doručení, musí aplikace běžící na NMS sama rozhodnout, zda si zažádat o zaslání informace znova. V případě trapu je problém, že NMS vůbec neví, že měl být doručen, takže v případě ztráty se o ničem nedoví. Může se zdát, že problém by vyřešilo použití TCP protokolu. Jelikož se SNMP začíná více používat až nastanou v síti problémy (pokud síť běží bez problému, není třeba nic diagnostikovat), tak by TCP ještě více zatěžovalo sít se žádostmi o znovu zaslání.

Pro zasílání dotazů a příjem odpovědí používá SNMP protokol UDP port 161 a pro příjem trapů port 162. Čísla portů mohou být samozřejmě změněny v nastavení aplikací, ale je třeba dbát tato nastavení provést jak u agenta, tak v NMS.

2.2.2 Komunity SNMP

V SNMP existují takzvané komunity, které představují omezení přístupových práv. V agentovi existují tři rozdílné úrovně komunit:

- read-only

  - uživatel může pouze číst data, nesmí je však modifikovat

- read-write

  - uživatel může jak číst data, tak i data modifikovat

- trap

  - uživatel může přijímat asynchronní zprávy trap

Jména úrovní komunit jsou vlastně hesla. Výrobci používají výchozí jméno *public* pro read-only komunitu a *private* pro read-write komunitu. Tyto hesla by měl každý uživatel změnit, ale velkou nevýhodou SNMPv1 nebo v2 je jejich zasílání v čistém textu, takže je potencionální útočník může snadno získat. Tento problém řeší až SNMPv3.

2.2.3 Struktura spravovaných informací (SMI)

Toto téma bylo již zlehka zmíněno v úvodu do SNMP protokolu. Je důležité vědět jak jsou data v SNMP protokolu reprezentována. Existují dvě verze SMIv1 a SMIv2.

Definice spravovaného objektu se dá rozdělit na 3 části:

- Jméno

  - Jméno nebo identifikátor objektu (OID – Object Identifier) jednoznačně určuje

spravovaný objekt. Obvykle je buďto číselné, nebo v lidsky čitelné podobě.

- Typ a syntaxe

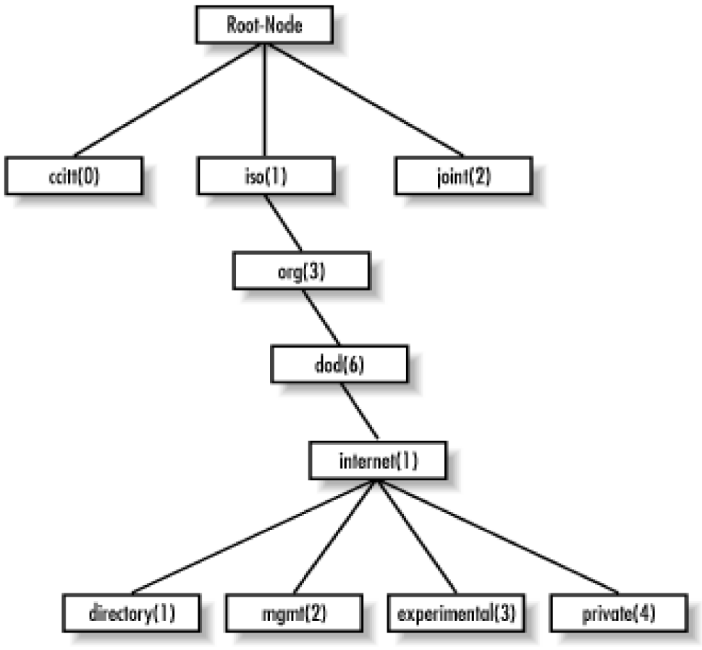
  - Specifikuje datový typ, jak jsou data reprezentovány a přenášeny mezi agenty a NMS. Ke specifikaci využívá podmnožinu Abstract Syntax Notation One (ASN.1)

- Kódování

  - Instance spravovaného objektu je zakódována do textového řetězce za použití základních kódovacích pravidel (BER – Basic Encoding Rules), následně může být objekt přenášen po transportním mediu.

2.2.3.1 Jména a OID

Spravované objekty tvoří stromovou strukturu, která tvoří základ systému pro pojmenování objektů v SNMP protokolu. Identifikátor objektu je tvořen sérií integerů založených na uzlech stromu oddělených tečkami, nebo sérií jmen odpovídající uzlům stromu také oddělených tečkami. Vše je patrné z obrázku 2, kde je ukázáno několik vrstev objektového stromu.



Obrázek 2: Objekotvý strom SMI [1]

Každý strom má ve svém vrcholu kořenový uzel (root). Každý uzel který má potomka se jmenuje podstrom a uzel bez potomka je nazýván list.

Jako ukázku můžeme použít podstrom iso(1).org(3).dod(6).internet(1) kde OID odpovídá 1.3.6.1 a textové jméno iso.org.dod.internet.

2.2.3.2 Definice objektů

Objekty jsou definovány pomocí podmnožiny syntaxe již zmiňované ASN.1. Ve verzi SMIv1je definováno několik datových typů, typy jsou obdobné jako v běžných programovacích jazycích. Datové typy jsou popsány v tabulce 1.

Datový typ	Popis
INTEGER	32bitové číslo, používaný často jako enumerátor
OCTET STRING	Určen pro text, nebo pro MAC adresy
Counter	Počítadlo od 0 do 2^{32}, sám se nuluje, může pouze zvyšovat svou hodnotu
OBJECT IDENTIFIER	Tečkami oddělený desítkový textový řetězec reprezentující objekt ve stromě (OID)
NULL	Nepoužíván.
SEQUENCE	Seznam obsahující žádný nebo více datových typů
SEQUENCE OF	Spravovaný objekt tvořený SEQUENCE
IpAddress	32bitová IPv4 adresa
NetworkAddress	Schodná s IpAddress, ale může reprezentovat jiný typ adresy
Gauge	Číslo od 0 do 2^{32} jde snižovat nebo zvyšovat, nikdy nepřeteče.
TimeTicks	Číslo od 0 do 2^{32} , které zaznamenává uběhlý čas ve stovkách milisekund
Opaque	Umožňuje jakémukoliv ASN.1 kódování být uloženo do OCTET STRING

Tabulka 1: Datové typy

Všechny tyto datové typy se využívají pro definici spravovaných objektů. Jak bylo napsáno, MIB shlukují spravované objety které se týkají jedné věci (zařízení, funkce,...) dohromady.

Ukázka MIB souboru (Výpis 1), řádky začínající -- jsou komentáře.

```
RFC1213-MIB DEFINITIONS ::= BEGIN

    IMPORTS
```

```
        mgmt, NetworkAddress, IpAddress, Counter, Gauge, TimeTicks
            FROM RFC1155-SMI
        OBJECT-TYPE
            FROM RFC 1212;

mib-2           OBJECT IDENTIFIER ::= { mgmt 1 }
system          OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces      OBJECT IDENTIFIER ::= { mib-2 2 }
at              OBJECT IDENTIFIER ::= { mib-2 3 }
ip              OBJECT IDENTIFIER ::= { mib-2 4 }
icmp            OBJECT IDENTIFIER ::= { mib-2 5 }
tcp             OBJECT IDENTIFIER ::= { mib-2 6 }
udp             OBJECT IDENTIFIER ::= { mib-2 7 }
egp             OBJECT IDENTIFIER ::= { mib-2 8 }
transmission    OBJECT IDENTIFIER ::= { mib-2 10 }
snmp            OBJECT IDENTIFIER ::= { mib-2 11 }

ifTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF IfEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
      "A list of interface entries. The number of entries is given by the value of ifNumber."
   ::= { interfaces 2 }

ifEntry OBJECT-TYPE
   SYNTAX  IfEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
```

```
        "An interface entry containing objects at the subnetwork layer and below for a particular
          interface."
    INDEX   { ifIndex }
    ::= { ifTable 1 }


IfEntry ::=
    SEQUENCE {
        ifIndex
            INTEGER,
        ifDescr
            DisplayString,
        ifType
            INTEGER,
        ifMtu
            INTEGER,
        ifSpeed
            Gauge,
        ifPhysAddress
            PhysAddress,
        ifAdminStatus
            INTEGER,
        ifOperStatus
            INTEGER,
        ifLastChange
            TimeTicks,
        ifInOctets
            Counter,
        ifInUcastPkts
            Counter,
        ifInNUcastPkts
```

```
            Counter,
        ifInDiscards
            Counter,
        ifInErrors
            Counter,
        ifInUnknownProtos
            Counter,
        ifOutOctets
            Counter,
        ifOutUcastPkts
            Counter,
        ifOutNUcastPkts
            Counter,
        ifOutDiscards
            Counter,
        ifOutErrors
            Counter,
        ifOutQLen
            Gauge,
        ifSpecific
            OBJECT IDENTIFIER
    }

ifIndex OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "A unique value for each interface. Its value ranges between 1 and the value of ifNumber. The
         value for each each interface must remain constant at least from one reinitialization of the
```

```
            entity's network-management  system to the next reinitialization."
        ::= { ifEntry 1 }


    ifDescr OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "A textual string containing information about the interface. This string should include the
            name of the manufacturer, the product name, and the version of the hardware interface."
        ::= { ifEntry 2 }
END
```

Výpis 1: MIB-II

První řádek obsahuje název MIB, který je: RFC1213-MIB. Sekce IMPORTS umožňuje importovat datové typy a OID z dalších MIB souborů. V našem případě se importuje

1 mgmt
2 NetworkAddress
3 IpAddress
4 Counter
5 Gauge
6 TimeTicks

z MIB s názvem RFC1155-SMI. Dále importujeme OBJECT-TYPE z MIB RFC-1212.

Následující sekce definuje mib-2 podstrom pomocí OID. Mib-2 je definována jako mgmt následováno .1. Mgmt je ekvivalent pro 1.3.6.1.2. Ekvivalent pro mib-2 tedy je 1.3.6.1.2.1. Ostatní části podstromu mib-2 jsou definovány obdobně. Například skupina interfaces je definována jako {mib-2 2} čemuž odpovídá 1.3.6.1.2.1.2.

Po definici všech OID následuje definice objektů. Každý objekt je tvořen danou strukturou (Výpis 2).

```
<name> OBJECT-TYPE
            SYNTAX <datatype>
            ACCESS <either read-only, read-write, write-only, or not-accessible>
```

```
        STATUS <either mandatory, optional, or obsolete>
        DESCRIPTION
                "Textual description describing this particular managed object."
        ::= { <Unique OID that defines this object> }
```

Výpis 2: Struktura MIB objektu

První spravovaný objekt v našem ukázkovém MIB souboru je objekt (Výpis 3) s jménem ifTable (jména objektů vždy začínají malým písmenem, ale zbytek jména může obsahovat malé i velké písmena), který odpovídá tabulce síťových rozhraní spravovaného zařízení.

```
ifTable OBJECT-TYPE
        SYNTAX SEQUENCE OF IfEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION
                "A list of interface entries. The number of entries is given by the value of ifNumber."
        ::= { interfaces 2 }
```

Výpis 3: Struktura MIB objektu ifTable

Objekt ifTable je definován jako SEQUENCE OF IfEntry. To znamená, že ifTable je tabulka obsahující sloupce definované v IfEntry. ACCESS not-accessible říká o objektu že je nepřístupný, takže se nelze dotazovat agenta na jeho hodnotu. STATUS mandatory určuje objektu že je povinný, čili agent ho musí implementovat v souladu se specifikací MIB-II. DESCRIPTION neboli popis slovně popisuje co objekt reprezentuje. Poslední informace je OID které odpovídá 1.3.6.1.2.1.2.2, neboli iso.org.dod.internet.mgmt.mib-2.interfaces.2.

Dále se je definována SEQUENCE IfEntry (Výpis 4), která je použita při definici SEQUENCE OF IfEntry v definici objektu ifTable.

```
IfEntry ::=
        SEQUENCE {
                ifIndex
                        INTEGER,
                ifDescr
                        DisplayString,
                ifType
```

```
                    INTEGER,
            ifMtu
                    INTEGER,
            . . .
            ifSpecific
                    OBJECT IDENTIFIER
}
```

Výpis 4: Struktura MIB sekvence IfEntry

SEQUENCE je jednoduše seznam sloupcových objektů a jejich datových typů. Na-rozdíl od objektů, jméno SEQUENCE začíná vždy velkým písmenem.

Informace v obsažené v celé této kapitole jsem čerpal z [1].

3 Net-SNMP projekt

Net-SNMP se zaobírá tvorbou a nasazením aplikací pro práci s SNMP protokolem. Je vyvíjen open source komunitou a jeho historie sahá až do roku 1992. Využívá se převážně na OS založených na OS Linux, ale existují i verze pro OS Windows.

3.1 SNMP server

Démon, takto je v prostředí OS Linux nazývaná aplikace běžící skrytě na pozadí, standartě plnící funkce SNMP serveru, nalezneme v OS Linuxu pod názvem *snmpd*. Jeho instalaci jsem provedl nainstalováním balíčku *snmpd* a v celé mé práci je využívána verze 5.7.2.

Po instalaci jsem musel provést základní nastavení serveru, jinak je funkčnost serveru výrazně omezena. Nastavení serveru je popsáno v manuálových stránkách nebo v [7].

3.2 Klienstké SNMP aplikace

Projekt vyvíjí několik aplikací pro příjem dat a další manipulaci s SNMP serverem.

- *snmptranslate*

  Překládá OID čísla na názvy a obráceně.

- *snmpget*

  Pošle požadavek na jednu konkrétní hodnotu OID ze serveru.

- *snmpgetnext*

  Podobný snmpget, akorát místo hodnoty OID vrátí další hodnotu za specifikovaným OID.

- *snmpwalk*

  Posílá požadavky na celý podstrom specifikovaného OID.

- *snmptable*

  Přečte obsah celé tabulky ze serveru.

- *snmpset*

  Nastaví hodnotu specifikovaného OID na novou.

- *snmpbulkget*

  Místo jednoho požadavku na jedno OID v jedné zprávě při snmpget nebo snmpgetnext, vyšle jednu zprávu pro více OID a odpověď je mu serverem odeslána opět v jedné zprávě.

- *snmpbulkwalk*

  Místo požadování podstromu OID po jednotlivých zprávách, pošle jednu hromadnou zprávu s a přijme hromadnou odpověď.

- *snmptrap*

  Pracuje s trapy. Jako klient můžu na serveru aktivovat trap, který následně server rozešle dalším odběratelům.

3.3 Vývoj pro Net-SNMP

Net-SNMP pro *snmpd* server umožňuje psát rozšiřující moduly za účelem přidání podpory nových MIB souborů. Pro vývoj jsem musel doinstalovat balík *snmpd-dev*, který zajistí potřebné hlavičkové soubory.

Nástroje:

- ***mib2c*** - Aplikace vytvoří jednoduchou šablonu v programovacím jazyce C s daty získanými z MIB souboru, namísto jejich složitého ručního kopírování.
- ***smilint*** - Pomocí aplikace jsem ověřil validitu mnou vytvořeného MIB souboru. Aplikace srozumitelně vypíše v čem a na kterém řádku je v MIB souboru problém.
- ***tkmib*** - Aplikaci jsem využil pro procházení MIB stromu v grafické podobě. Aplikace usnadní a zpřehlední závislosti a významy položek v MIB stromu.

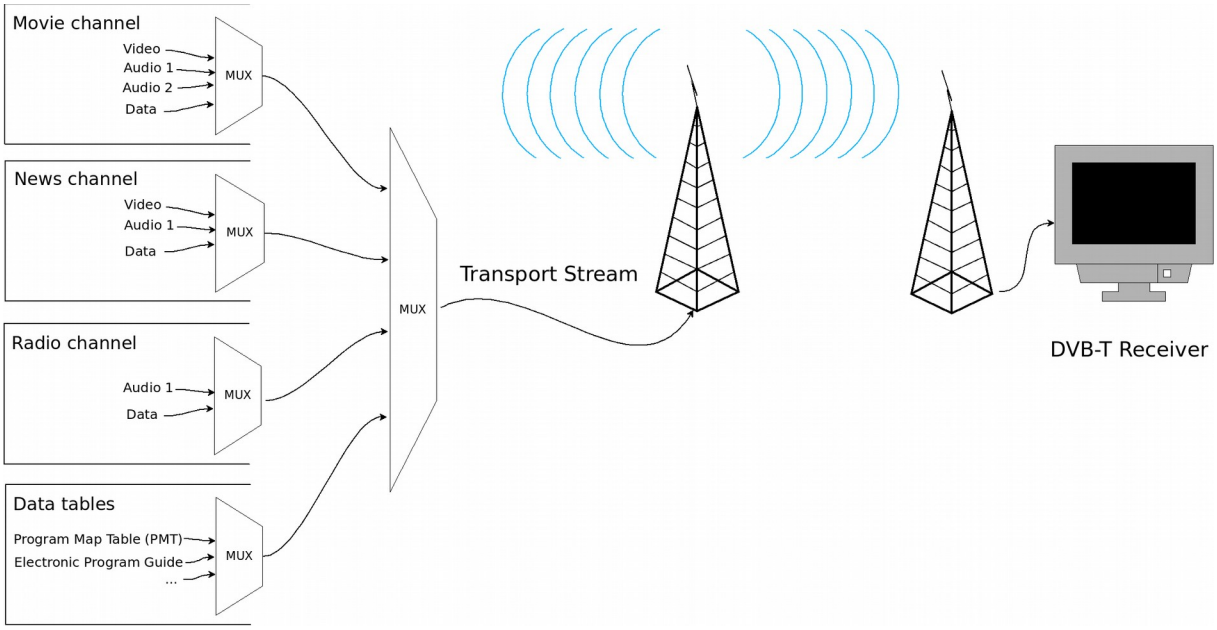
Bližší informace, podrobnější nastavení a další možnosti pro kapitolu 3 jsou k nalezení v [7].

4 DVB vysílání

DVB obsahuje celosvětově uznávané standarty sloužící k šíření videa v digitální podobě. V současnosti existuje několik specifikací DVB v závislosti na způsobu šíření signálu, mezi nejznámější z nich patří:

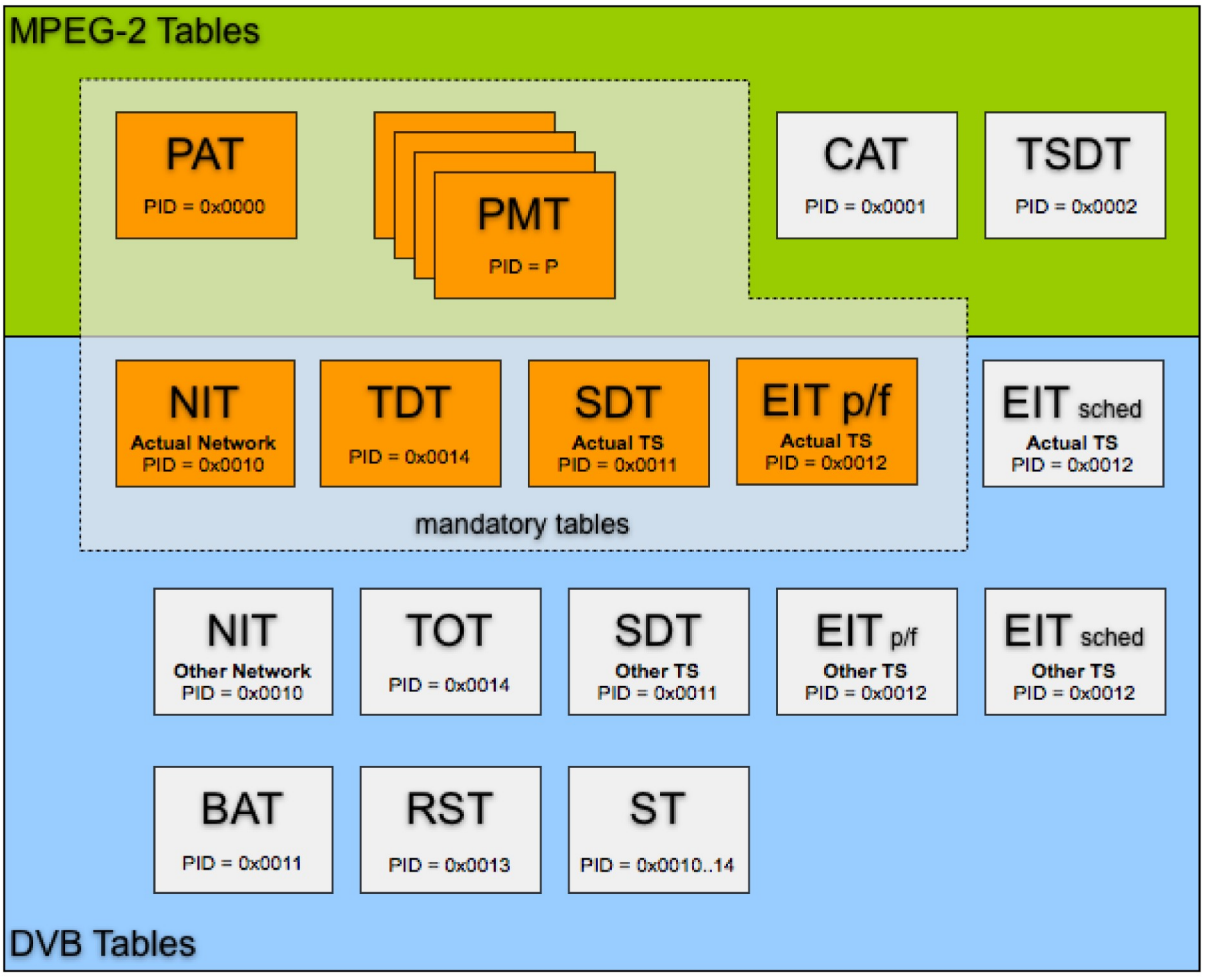
- DVB-T - Pozemní digitální vysílání.
- DVB-S - Satelitní digitální vysílání.
- DVB-C - Kabelové digitální vysílání.

Základní myšlenkou digitálního šíření je nabídnout několik služeb (nemusí být pouze pouze video) na jedné nosné frekvenci, čímž lépe využívá frekvenční spektrum oproti analogovému vysílání, kdy jednu nosnou frekvenci celou využívá pouze jedna služba. Princip je ukázán na obrázku 3.



Obrázek 3: Princip DVB vysílání

Pro další práci mě bude převážně zajímat část datových tabulek, které obsahují pro mě potřebné informace. DVB využívá pro přenos dat standardizovaný kontejner MPEG Transport Stream, který definuje jako takzvané PSI čtyři informační tabulky (PAT, CAT, PMT, TSDT) a DVB přidává jako SI dvanáct dalších informačních tabulek (NIT, SDT, EIT, TDT ….). Podrobný rozpis PSI/SI tabulek je znázorněn na obrázku 4.



Obrázek 4: PSI/SI tabulky[4]

Každý druh tabulky je identifikován ve vysílaném paketu číslem PID, jakožto i každý jednotlivý proud. Tabulka má největší možnou velikost 4096 bajtů (=sekce) a pro přenos je rozdělena do paketů s velikostí pro DVB-T/S/C 204 bajtů.

Každá tabulka obsahuje pevně definované části plus může obsahovat v určených místech smyčky takzvaných deskriptorů, což se dá chápat jako pod-tabulka. V desktriptorové smyčce může být nula až N deskriptorů a nemusí být stejného typu.

Podrobný popis struktury PSI/SI tabulek je uveden v dokumentech [2] a [4], které jsou jim velmi podrobně věnovány.

5 DVB zařízení v OS Linux

Kapitola se věnuje základní problematice DVB zařízení v OS Linux. Tyto informace byly pro mě důležité pro další pokračováni práce.

5.1 Instalace DVB zařízení

Abych mohl používat DVB přijímač na počítači s OS Linux, je potřeba správné volby zařízení z důvodu dostupnosti ovladače.

Zda je zřízení systémem rozpoznáno je signalizováno zobrazením odkazů na jednotlivé moduly zařízení (frontend, demux....) do složky /dev/dvb/. Jednotlivá DVB zařízení jsou pak v podsložkách *adapterN*, kde N je číslo adaptéru. Běžně je maximální počet DVB adaptérů omezen na 8 (čili N=0 až N=7).

Průběh inicializace (Výpis 5) úspěšně rozpoznaného DVB zařízení získané příkazem *dmesg*:

```
[   15.166643] usb 2-1.5: dvb_usb_v2: found a 'E3C EC168 reference design' in cold state
[   15.274288] usb 2-1.5: dvb_usb_v2: downloading firmware from file 'dvb-usb-ec168.fw'
[   15.339063] usb 2-1.5: dvb_usb_v2: found a 'E3C EC168 reference design' in warm state
[   15.339117] usb 2-1.5: dvb_usb_v2: will pass the complete MPEG2 transport stream to the software demuxer
[   15.339143] DVB: registering new adapter (E3C EC168 reference design)
[   15.386710] usb 2-1.5: DVB: registering adapter 1 frontend 0 (E3C EC100 DVB-T)...
[   15.454401] MXL5005S: Attached at address 0xc6
[   15.454405] usb 2-1.5: dvb_usb_v2: 'E3C EC168 reference design' successfully initialized and connected
[   15.454422] usbcore: registered new interface driver dvb_usb_ec168
```

Výpis 5: dmesg

5.2 Struktura DVB zařízení

Složka *adapterN* typicky obsahuje několik modulů DVB zařízení:

- **frontend** - Ovládá tuner (vybírá přijímanou frekvenci) a demodulátor. Tyto zařízení mají různé parametry, podle typu přijímaného DVB signálu (DVB-T, DVB-S atd.).
- **demux** - Souží jako filtr pro TS. Rozdělí proud na jednotlivé části, audio proud, video proud, případně datové proudy. Na zařízení se lze připojit více než jednou a tedy aktivovat více různých filtrů.

- **dvr** - Sestavuje nový TS na základě selekce z několika proudů původního hlavního TS, získaných pomocí demux zařízení. Na dvr je možné pouze jedno jedno připojení.

5.3 Použití DVB zařízení

Abych mohl monitorovat přijímaný signál je zapotřebí nejprve zjistit dostupné vysílací frekvence (kanály) a následně z jedné zvolené přijímat službu.

Kanály jsem vyhledával pomocí programu *scan*, případně jeho modifikací, podle potřebných požadavků a technologie příjmu (DVB-T, DVB-S, atd.) Tyto programy jsou součástí balíku aplikací *dvb-apps*.

Následně musím jeden z nalezených kanálu zvolit a začít přijímat. Toto provedu například aplikací *zap* a jejími modifikacemi z balíku dvb-apps, případně mohu využít některou z grafických aplikaci určenou pro příjem televizního vysílání (*mplayer*, *vlcplayer*, *me-tv*, atd.).

Nyní už mám zajištěn příjem signálu, jehož vybrané části mohu v dalších částech práce přijímat z demux zařízení a monitorovat.

Kapitola 5 čerpá převážně z webových stránek zabývajících se příjmem televize v OS Linux od autorů DVB API a balíku *dvb-apps* [5].

6 Analýza MIB souborů

DVB konsorcium ve spolupráci s ETSI definovali v dokumentu [3] celkem tři MIB soubory určené pro měření a monitorování DVB systémů. Soubory mají následují názvy:

- DVB-MGSYSTEM-MIB
- DVB-MGSIGNALCHARACTERISTICS-MIB
- DVB-MGTR101290-MIB

6.1 DVB-MGSYSTEM-MIB

Tento rozsahem nejkratší soubor obsahuje informace o názvu, poloze, správci a dalších základních informacích o serveru. Tyto informace se taktéž dají nalézt v MIB-II souboru, který musí implementovat každý SNMP server. Z hlediska monitorování DVB je tedy tento soubor nevýznamný.

6.2 DVB-MGSIGNALCHARACTERISTICS-MIB

V tomto souboru se nacházejí převážně informace týkající se transportního streamu (TS) a informací o něm nacházející se v takzvané SI části proudu. To znamená, že obsahují informace o aktuálním multiplexu jako je počet kanálů, jejich typu, přítomnosti teletextu, šifrování a dalších. Tyto informace jsou součástí informačního proudu každého DVB vysílání, takže se dají odchytit a při znalosti jejich struktury i dekódovat. Srovnáním možností DVB API a MIB souboru jsem vybral následující tabulky, pro které jsem schopen získat odpovídající hodnoty.

- mgTSTable
- mgServiceTable
- mgPIDTable
- mgEMMTable
- mgServiceECMTable
- mgPIDECMTable
- mgNITDeliverySystemTable

6.2.1 mgTSTable

Obsahuje informace (Tabulka 2) týkající se TS.

Název:	Popis:
mgTSInputNumber*	Číslo vstupu (DVB adaptéru), na kterém je tento TS přijímán.
mgTSId	Identifikátor TS, který nalezneme v PAT tabulce jako transport_stream_id.
mgTSOriginalNetworkID	Identifikátor TS který nalezneme v transport_stream_loop pro aktuální TS v NIT tabulce pod názvem original_network_id.
mgTSNetworkID	Identifikátor TS který nalezneme jako network_id v NIT tabulce (pro tento proud).
mgTSNetworkName	Jméno pro tento TS zapsané v NIT tabulce (pro aktuální proud) po dekódování name deskriptoru v položce network_name.

Tabulka 2: mgTSTable

*položky sloužící v tabulce jako primární klíč

6.2.2 mgServiceTable

Tabulka obsahuje informace (Tabulka 3) o službách poskytovaných v aktuální službě. Služby jsou skutečně vysílány v tomto proudu pokud jsou zahrnuty jak v PAT tak i PMT tabulce.

Název:	Popis:
mgServiceInputNumber*	Číslo vstupu (DVB adaptéru), na kterém je tento TS přijímán.
mgServiceNumber*	Číslo služby service_id nacházející se v SDT tabulce v části service_descriptor.
mgServiceType	Typ služby service_type nacházející se v SDT tabulce v části service_descriptor.
mgServiceName	Jméno služby service_name nacházející se v SDT tabulce v části service_descriptor.
mgServiceProviderName	Jméno poskytovateke služby service_provider_name nacházející se v SDT tabulce v části service_descriptor.
mgServicePMTPID	Číslo PID, pod kterým se vysílá příslušná PMT tabulka k této službě. Nachází se v PAT tabulce.
mgServicePCRPID	Číslo PID, na kterém je vysílán packet s PCR pro tuto službu. Nachází se v PMT tabulce.
mgServiceCondAccess	Identifikátor omezení přístupu free_CA_mode nacházející se v SDT tabulce v části service_descriptor.
mgServiceEITComponentDescriptor	Textový popis hlavního proudu v této službě. Nachází se v

	EIT tabulce v deskriptoru component_descriptor. Při implementaci této položky jsem narazil na problém velké variability a volnosti obsahu EIT tabulek. Tedy jsem nebyl schopen získat validní zdroj dat pro tuto položku a je naplněna ve všech případech prázdným textovým řetězcem. Naplnění prázdným textovým řetězcem je povoleno v MIB specifikaci u této položky v případě její nedostupnosti a tudíž nedochází k rozporu s MIB souborem.

Tabulka 3: mgServiceTable

*položky sloužící v tabulce jako primární klíč

6.2.3 mgPIDTable

Tabulka obsahuje informace (Tabulka 4) o všech PID číslech, které mají souvislost s každou službou dostupnou v TS.

Název:	**Popis:**
mgPIDInputNumber*	Číslo vstupu (DVB adaptéru), na kterém je tento TS přijímán.
mgPIDServiceNumber*	Číslo služby service_id nacházející se v SDT tabulce v části service_descriptor.
mgPIDNumber*	PID číslo související se službou identifikované v mgPIDServiceNumber jako položka elementary_PID v deskriptorové smyčce PMT tabulky.
mgPIDType	Typ streamu nacházející se jako stream_type v deskriptorové smyčce PMT tabulky.
mgPIDCondAccess	Stav šifrování u vysílaného proudu s výše zmíněným PID číslem. Získá se z položky transport_scrambling_control v hlavičce TS paketu. K tomu abych získal hlavičku paketu TS musím být připojen na dvr zařízení, ale protože DVB přijímač již přijímá signál prostřednictvím jiného procesu a tudíž je dvr část používána a neumožňuje více než jedno připojení. Připojením na zařízení demux získám pouze pakety zbavené hlavičky. Z těchto důvodů je hodnota této položky nastavena vždy na neurčeno.

Tabulka 4: mgPIDTable

*položky sloužící v tabulce jako primární klíč

6.2.4 mgEMMTable

Tabulka obsahuje informace (Tabulka 5) o číslech PID na kterých se vysílají EMM zprávy.

Název:	**Popis:**
mgEMMInputNumber*	Číslo vstupu (DVB adaptéru), na kterém je tento TS přijímán.
mgEMMCaPID*	Číslo PID na kterém je vysílána EMM zpráva. Nachází se v CA_descriptor jako položka CA_PID v CAT tabulce.
mgEMMCASystemID	Identifikátor poskytovatele služby s podmíněným přístupem. Nachází se v CA_descriptor jako položka CA_system_ID v CAT tabulce.

Tabulka 5: mgEMMTable

*položky sloužící v tabulce jako primární klíč

6.2.5 mgServiceECMTable

Tabulka obsahuje PID čísla související s ECM zprávami pro celou službu (Tabulka 6).

Název:	**Popis:**
mgServiceECMInputNumber*	Číslo vstupu (DVB adaptéru), na kterém je tento TS přijímán.
mgServiceECMServiceNumber*	Identifikátor služby. Najdeme ho v PMT tabulce pod názvem service_id.
mgServiceECMCaPID	Číslo PID na kterém se vysílá ECM zpráva. Nachází se v CA_descrtiptor v PMT tabulce pod názvem CA_PID.
mgServiceECMCASystemID	Identifikátor poskytovatele služby s podmíněným přístupem. Nachází se v CA_descriptor jako položka CA_system_ID v PMT tabulce.

Tabulka 6: mgServiceECMTable

6.2.6 *položky sloužící v tabulce jako primární klíč

6.2.7 mgPIDECMTable

Tabulka obsahuje PID čísla související s ECM zprávami pro každý proud ve službě (Tabulka 7).

Název:	**Popis:**
mgPIDECMInputNumber*	Číslo vstupu (DVB adaptéru), na kterém je tento TS přijímán.
mgPIDECMServiceNumber*	Identifikátor služby. Najdeme ho v PMT tabulce pod názvem service_id.

mgPIDECMPID*	Číslo PID na kterém se vysílá tento konkrétní proud ze služby. Proud s tímto PID je dešifrován ECM zprávou z PID mgPIDECMCaPID.
mgPIDECMCaPID	Číslo PID na kterém se vysílá ECM zpráva. Nachází se v CA_descrtiptor v PMT tabulce pod názvem CA_PID.
mgPIDECMCASystemID	Identifikátor poskytovatele služby s podmíněným přístupem. Nachází se v CA_descriptor jako položka CA_system_ID v PMT tabulce.

Tabulka 7: mgPIDECMTable

*položky sloužící v tabulce jako primární klíč

6.2.8 mgNITDeliverySystemTable

Tabulka obsahuje informace (Tabulka 8) o parametrech šíření TS. Veškeré informace pro tuto tabulku se nacházejí v NIT tabulce.

Název:	**Popis:**
mgNITDSInputNumber*	Číslo vstupu (DVB adaptéru), na kterém je tento TS přijímán.
mgNITDSSystemType	Nalezneme v položce tag v delivery_system_descriptoru. Na základě typu sytému se mění i položky ve zbytku deskriptoru a deskriptor obsahuje pouze některé informace, které souvisí s typem sytému.
mgNITDSFrequency	Pokud je v delivery_system_descriptoru položka centre_frequency nebo frequency, tak je to její hodnota jinak 0.
mgNITDSFecOuter	Pokud je v delivery_system_descriptoru položka fec_outer, tak je to její hodnota jinak 0.
mgNITDSCableModulation	Pokud je v delivery_system_descriptoru položka modulation, tak je to její hodnota jinak 0.
mgNITDSSymbolRate	Pokud je v delivery_system_descriptoru položka symbol_rate, tak je to její hodnota jinak 0.
mgNITDSFecInner	Pokud je v delivery_system_descriptoru položka fec_inner, tak je to její hodnota, jinak 0.
mgNITDSOrbitalPosition	Pokud je v delivery_system_descriptoru položka orbital_position, tak je to její hodnota jinak 0.
mgNITDSWestEastFlag	Pokud je v delivery_system_descriptoru položka

	west_east_flag, tak je to její hodnota jinak 0.
mgNITDSPolarization	Pokud je v delivery_system_descriptoru položka polarization, tak je to její hodnota jinak 0.
mgNITDSSatelliteModulation	Pokud je v delivery_system_descriptoru položka modulation_type, tak je to její hodnota jinak 0.
mgNITDSBandwidth	Pokud je v delivery_system_descriptoru položka bandwidth, tak je to její hodnota jinak 0.
mgNITDSConstellation	Pokud je v delivery_system_descriptoru položka constellation, tak je to její hodnota jinak 0.
mgNITDSHierarchyInformation	Pokud je v delivery_system_descriptoru položka centre_frequency, tak je to její hodnota jinak 0.
mgNITDSCodeRateHPStream	Pokud je v delivery_system_descriptoru položka code_rate_hp_stream, tak je to její hodnota jinak 0.
mgNITDSCodeRateLPStream	Pokud je v delivery_system_descriptoru položka code_rate_lp_stream, tak je to její hodnota jinak 0.
mgNITDSGuardInterval	Pokud je v delivery_system_descriptoru položka guard_interval, tak je to její hodnota jinak 0.
mgNITDSTransmissionMode	Pokud je v delivery_system_descriptoru položka transmission_mode, tak je to její hodnota jinak 0.
mgNITDSOtherFrequencyFlag	Pokud je v delivery_system_descriptoru položka other_frequency_flag, tak je to její hodnota jinak 0.

Tabulka 8: mgNITDeliverySystemTable

*položky sloužící v tabulce jako primární klíč

6.3 DVB-MGTR101290-MIB

Poslední a nejrozsáhlejší soubor obsahuje tabulky s výsledky mnoha specifických měření a testů kvality signálu a přenosu. Na většinu z nich nelze použitím prostředků DVB API získat korektní odpověď a je zapotřebí mít sofistikované měřící přístroje. Z mého pohledu jsou tyto informace již nedůležité a příliš pokročilé pro potřeby monitorování provozu vysílacího serveru jako takového a jsou určeny pro osoby zabývající se DVB vysíláním a měřením kvality jeho pokrytí.

6.4 Vlastní MIB soubor DVB-FRONTENDCHARACTERISTICS-MIB

Jelikož se dá získat několik informací, které dle mě mají význam pro monitorování vysílacího serveru, ale nejsou v takto jednoduché formě obsaženy v žádném z oficiálně definovaných MIB souborů, vytvořil jsem vlastní MIB soubor, obsahující jednu tabulku:

- mgDCHTable

6.4.1 mgDCHTable

Tabulka shromažďuje dynamická data z frontend části DVB přijímače. Jedná se například o kvalitu signálu, chybovost atd (Tabulka 9).

Název:	**Popis:**
mgDCHInputNumber*	Číslo vstupu (DVB adaptéru), na kterém je tento TS přijímán.
mgDCHSignalStrength	Síla signálu v procentech. Získám zavoláním *ioctl* s parametrem FE_READ_SIGNAL_STRENGTH nad vybraným frontend zařízením.
mgDCHSignalBER	Síla signálu v procentech. Získám zavoláním *ioctl* s parametrem FE_READ_BER nad vybraným frontend zařízením.
mgDCHSignalSNR	Síla signálu v procentech. Získám zavoláním *ioctl* s parametrem FE_READ_SNR nad vybraným frontend zařízením.
mgDCHSignalUncorrectedBlocks	Síla signálu v procentech. Získám zavoláním *ioctl* s parametrem FE_READ_UNCORRECTED_BLOCKS nad vybraným frontend zařízením.
mgDCHSignalFrequency	Aktuálně nalazená střední frekvence frontend zařízení. Získám zavoláním *ioctl* s parametrem FE_GET_FRONTEND nad vybraným frontend zařízením a následně výběrem položky frequency ze získané struktury.

Tabulka 9: mgDCHTable

*položky sloužící v tabulce jako primární klíč

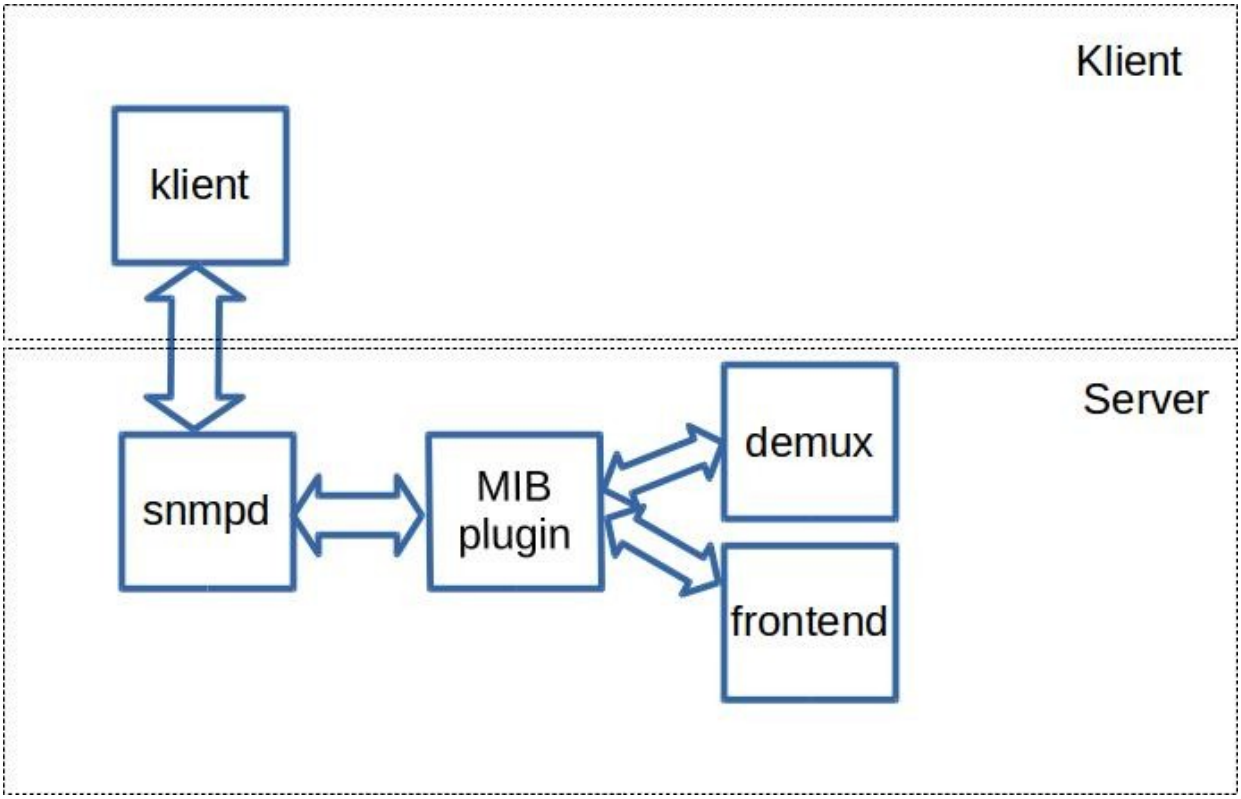
Informace pro kapitolu 6 jsem čerpal převážně z dokumentu [2] a webových stránek věnovaných DVB zařízením v OS Linux [5]. Další podklady jsem také nalezl v popisovaných MIB souborech a dokumentu jim věnovaným [3].

7 Návrhy řešení

Po předchozí analýze MIB souborů jsem vytvořil tři možné návrhy řešení, mezi nimiž jsem se musel a rozhodnout a vybrat ten nejvíce vhodný.

1. Vše obstarat pouze v MIB modulech v *snmpd* serveru.
2. Udělat aplikaci pro sběr dat a v MIB modulech následně pouze hodnoty načítat a odesílat klientům.
3. Kombinace předchozích dvou.

7.1 Řešení pouze v MIB modulech



Obrázek 5: Řešení pouze s MIB moduly

Myšlenka byla na první pohled jednoduchá. Při požadavku na data z MIB tabulky, SNMP server jednoduše zavolá modul odpovědný pro získání těchto dat. Modul se posléze připojí na frontend nebo demux zařízení, odchytí požadované pakety, nebo vyzíská údaje. Získaná data následně zpracuje, předá SNMP serveru a odešle v odpovědi klientu. Grafické znázornění na obrázku 5.

Toto řešení bohužel není nejlepší, vyskytuje se zde problém délky zpracování. Pokud požadujeme něco z MIB souboru obsahující informace z PSI/SI tabulek DVB, trvá jejich zachycení a zpracování relativně dlouhou dobu. Když přijde další požadavek na jiná data z MIB souboru opět z PSI/SI tabulek DVB, musí se celá procedura provést znovu, když už byla částečně provedena předtím, jelikož se obsah MIB tabulek prolíná.

Dalším faktorem dlouhé doby vykonávání je, že PSI/SI tabulky se vysílají v určitých intervalech a tedy určitou dobu trvá, než bude potřebná tabulka zachycena. Doba na odpověď serveru je omezena a v některých případech by mohlo docházet k vypršení časového limitu pro odpověď.

Toto řešení je vhodné pro dynamická data získávána z frontend zařízení, která ale tvoří pouze malou část v MIB souborech.

Po této úvaze jsem shledal tento typ řešení jako nevhodné.

7.2 Řešení se sběrem dat

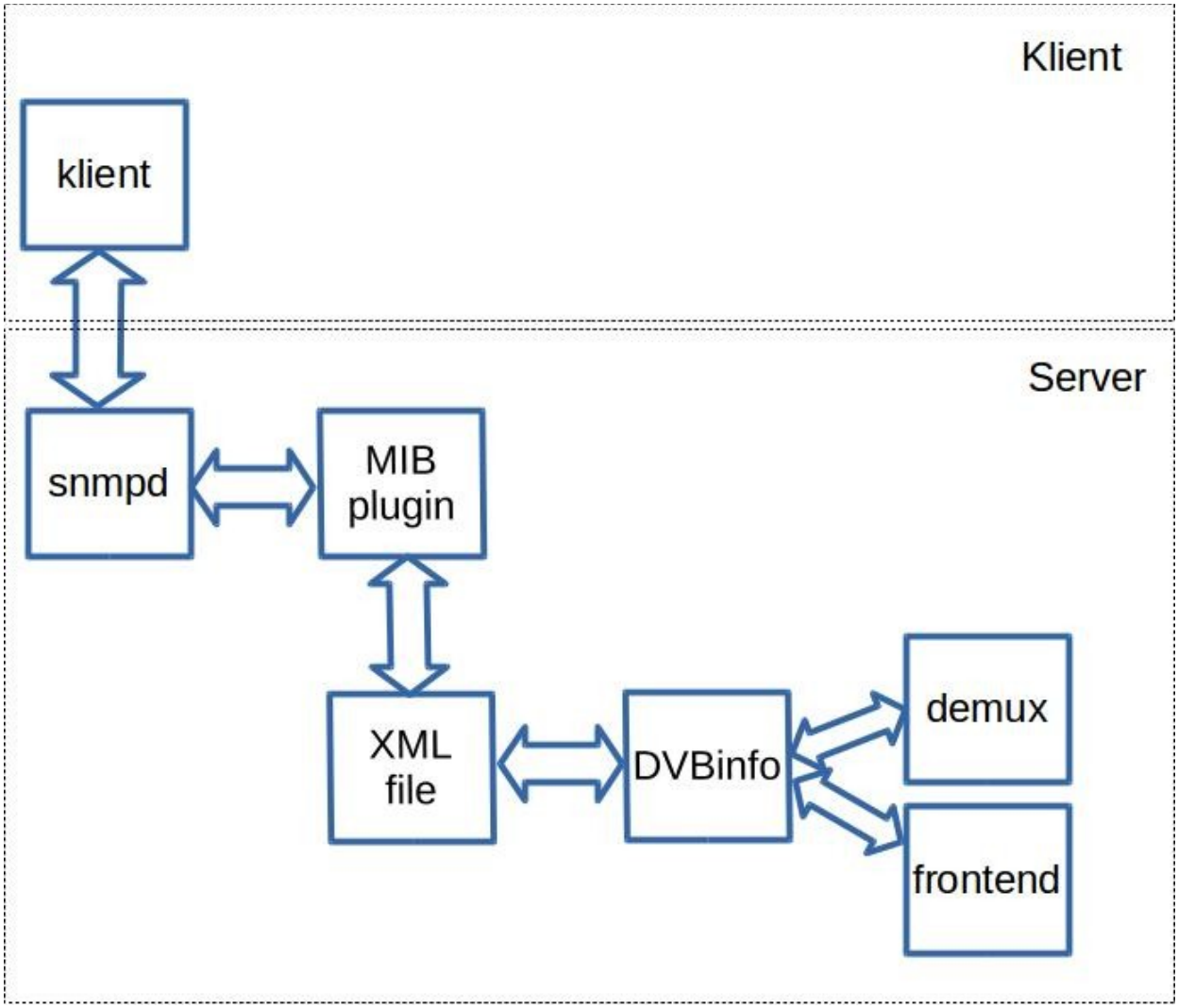
Řešení spočívá ve vytvoření aplikace (DVBinfo), která sbírá potřebná data a ukládá je ve formátu XML souboru. Při požadavku na data z MIB souboru SNMP server zavolá modul určený pro tyto data, kde je modul načte z XML souboru. Princip je graficky znázorněn na na obrázku 6.

V tomto případě nevzniká žádná časová prodleva a odpovědi SNMP serveru jsou téměř okamžité. Čas pouze zabírá načítání dat z XML souboru. S příchodem XML souboru jako mezi článku vzniká problém s přístupem k jednomu souboru ze dvou různým aplikací. Toto se dá vyřešit zámky na soubor, a ty když jsou vhodně v programu umístěny, tak čekání na odpověď se v případě kolize prodlouží jen nepatrně.

Problém nastává v kombinaci poměrně statických a neněmých dat z PSI/SI a dynamicky měnících se dat z frontend zařízení. Je totiž zbytečné s velkou periodicitou neustále obnovovat téměř statická data, a na druhou stranu obnovování dynamických dat bude zpomalováno dlouhým vykonáváním zpracovávání statických dat.

Řešením by mohlo být vytvořit dvě aplikace a dva XML soubory, s níž každá by se spouštěla s jiným intervalem. Ale na druhou stranu je zbytečné, aby na pozadí běžely dvě aplikace, které spotřebují zbytečně 2x více systémových prostředků než jedna. Aplikace pro získávání dynamických dat by musela běžet prakticky nepřetržitě, aby XML soubor měl stále aktuální data, čímž by rapidně narůstalo riziko kolize v přístupu k XML souboru. A také když nikdo dlouho nepošle požadavek na tato data, tak aplikace bude úplně zbytečně vytěžovat server. Na druhou stranu aplikace pro získání neměnných dat by se spouštěla v relativně dlouhém intervalu několikrát za den, což je optimální řešení.

Takže druhé řešení i s vylepšením dvěma aplikacemi není pořád ideální a nabízí se tedy možnost kombinace prvního a druhého řešení, ve kterých je vždy jedna část vyřešena přijatelně a druhá ne.



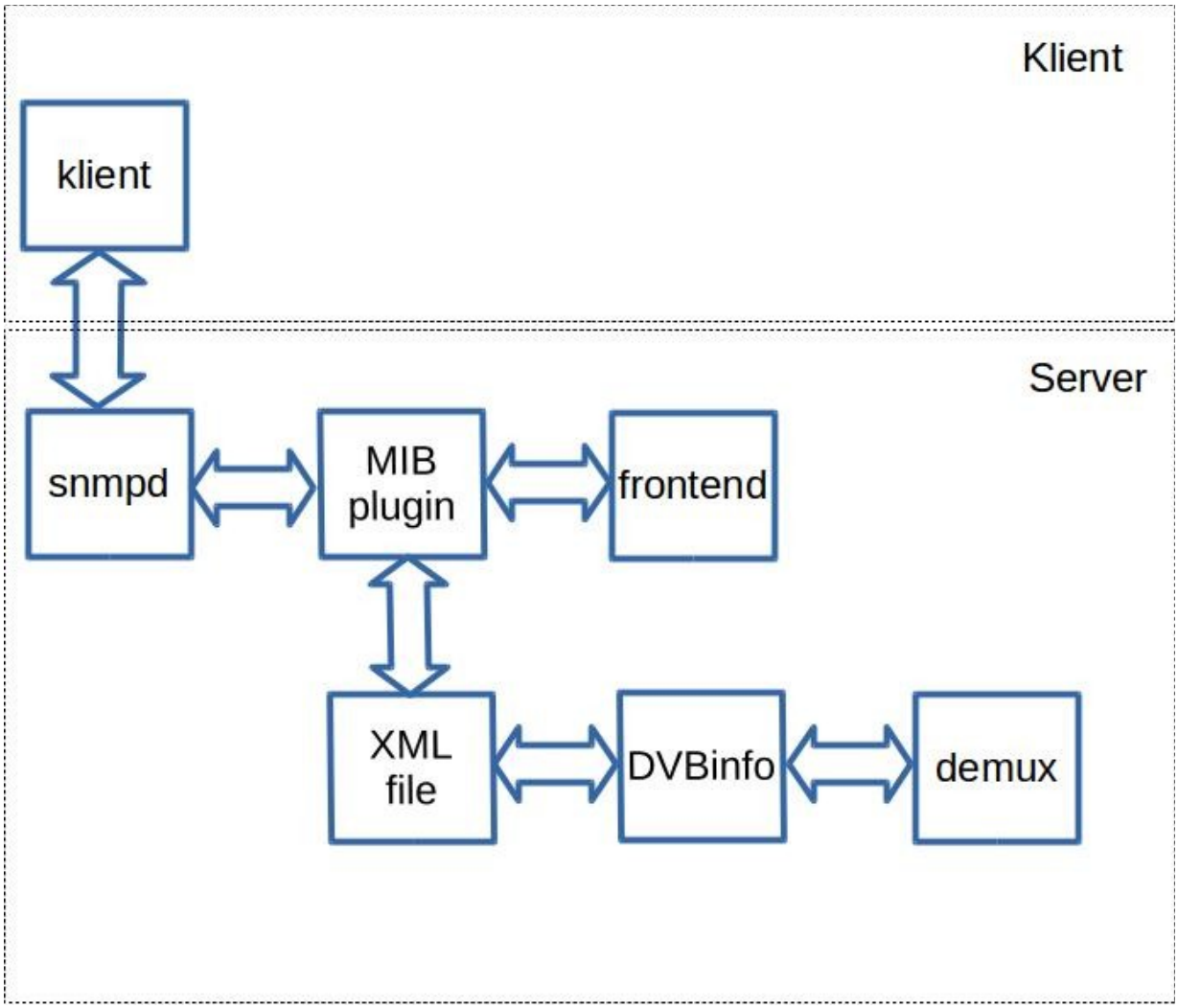
Obrázek 6: Řešení s použitím XML souboru

7.3 Řešení kombinací sběru dat a pouze MIB moduly

Použité řešení vychází jako kombinace předchozích dvou řešení. Vytvořil jsem aplikaci DVBinfo odchytávající a zpracovávající PSI/SI proud z DVB vysílaní, který ukládá data v XML souboru. Pro *smpdp* server jsem vytvořil potřebné moduly, které načítají data z XML souboru. Aplikace DVBinfo bude v tomto případě spouštěna v intervalu několikrát za den a nebude tedy příliš zatěžovat svým během výkon serveru.

Ovšem s přístupem ke sdílenému XML souboru vzniká opět problém s možností kolize přístupu. I když tato pravděpodobnost bude velice malá, z důvodů malého počtu přístupů k souboru z aplikace DVBinfo, ale není nezanedbatelná a musí se proto v aplikaci a MIB modulech ošetřit.

Dynamická data načítaná z frontendu se získávají přímo z *snmpd* modulů, takže při požadavku na tyto data SNMP server vrací téměř aktuální hodnotu. Princip je graficky znázorněn na obrázku 7.



Obrázek 7: Řešení kombinací sběru dat a pouze MIB modulů

Toto řešení mi přišlo s ohledem na efektivitu a funkčnost jako nejlepší a bylo tedy použito pro realizaci projektu.

8 Aplikace DVBinfo

Jako první jsem začal práci na tvorbě aplikace DVBinfo. Aplikace přijímá PSI/SI tabulky z TS, dekóduje je a ukládá hodnoty do XML souboru v takovém formátu, aby odpovídal požadavkům na data definované v MIB souboru.

8.1 Základní funkce a použití

Aplikace DVBinfo odchytává postupně PSI/SI tabulky ze všech dostupných DVB adaptérů, získává data potřebné k implementaci MIB souborů a tyto data ukládá v XML souboru.

Aplikace je součástí instalačního balíku *dvbsnmp*, po jehož instalaci ji bude uživatel moci pouštět příkazem *dvbinfo* v terminálu. Z důvodů automatizace získávání dat je během instalace nastavena cron úloha, která spouští tuto aplikaci automaticky každou hodinu. Výsledný XML soubor uživatel nalezne ve složce /tmp/dvbinfoout.xml.

V případě nutnosti dřívější aktualizace ji uživatel může spustit ručně příkazem *dvbinfo*, pro případ potřeby lazení je možné aplikaci spustit s parametrem *dvbsnmp -d* kdy je do terminálu vypsán přehledný výstup a průběh zpracování tabulek. Při normální spuštění vypisuje aplikace pouze závažné chyby. Aplikaci je možné spustit také s parametrem *dvbsnmp -h*, který vypíše stručnou nápovědu.

8.2 Použité nástroje a technologie

- Aplikace je psaná v programovacím jazyce C++.
- V aplikaci využívám služeb knihovny *libucsi* která je součástí balíku *dvb-apps* a je k určena k dekódovaní přijatých rámců TS a tedy i rámců s PSI/SI.
- Pro práci s XML využívám C++ knihovnu pugiXML.
- Aplikaci vyvíjím ve vývojovém prostředí Netbeans.
- Nástoj *valgrind* pro ověření možných úniků paměti.

8.3 Ukázky zdrojového kódu

Ukázky celých funkcí, nebo vybraných části zdrojového kódu, které objasňují a prezentují princip fungování aplikace DVBinfo.

8.3.1 Hlavní funkce

Hlavní funkce celého programu (Zdrojový kód 1). Na počátku je definice a inicializace potřebných proměnných pro zjištění průchodu všemi potenciálně dostupnými DVB zařízeními a vytvořen XML

dokument pro uložení získaných dat. Cyklus while se postupně pokusí projít a připojit se na všechny potencionálně dostupné demux zařízení. Cesty ke všem potencionálně existujícím demux zařízením se důmyslně tvoří ve funkci *snprintf*, kdy jsou do textového řetězce /dev/dvb/adapter%d/demux%d postupně v cyklu dosazovány všechny jejich možné čísla, kdy následně ověřím zda takovýto demux existuje. V případě existence demuxu, se na něho připojím prostřednictvím funkce *openDemux*. Po úspěšném připojení se vykoná funkce *processTables* s tímto demuxem. Po skončení cyklu se uloží XML dokument se získanými daty.

```
#define MAXDEVICECOUNT 8
#define DEMUXDEVICE "/dev/dvb/adapter%d/demux%d"
int process(){
    int ret = 1;
    int demux = 0;
    int adapter = 0;
    int inputNumber = 1; //identificator of demux
    char dedev[128]; //contains actual path to demux device
    pugi::xml_document *doc = new pugi::xml_document;

    loadXMLDocument(doc);
    while (inputNumber <= MAXDEVICECOUNT) {
        snprintf(dedev, sizeof (dedev), DEMUXDEVICE, adapter, demux);
        printf("Try to open device: %s\n", dedev);
        if (access(dedev, F_OK) != -1) {
            printf("Success, device exist!!! \n");
            printf("Try to get connection \n");
            int defd;
            if((defd = openDemux(dedev)) != 0){
                printf("Connected to demux device.\n");
                if(processTables(defd, inputNumber, doc) == 0){
                    ret = 0;
                }
                closeDemux(defd);
```

```
            }else {
                printf("Connection to demux failed.\n");
            }
        } else {
            printf("Fail, device doesn't exist!!! \n");
        }
        adapter++;
        inputNumber++;
    }

    saveXMLDocument(doc);
    delete doc;
    return ret;
}
```

Zdrojový kód 1: void process()

8.3.2 Funkce pro připojení se k demux zařízení

Funkce *openDemux (Zdrojový kód 2)* má jedinný vstupní paremetr, kterým je cesta k požadovanému demux zařízení. Pomocí funkce *open* se pokusí připojit na demux. Demux se otevře pro čtení a zápis (O_RDWR) a jako velký soubor (O_LARGEFILE). Pokud nedojde k chybě, je vrácen deskriptor na demux (kladné celé číslo). V případě chyby je vrácena 0.

```
int openDemux(char * dedev) {
    int defd;

    if ((defd = open(dedev, O_RDWR | O_LARGEFILE)) < 0) {
        perror("Opening demux failed");
        return 0;
    }
    return defd;
}
```

Zdrojový kód 2: int openDemux()

8.3.3 Funkce pro načtení rámce se specifikovaným PID číslem

Funkce *readPid (Zdrojový kód 3)* požaduje jako vstupní parametry deskriptor demux zařízení, pole pro nahrání přijatých dat s jeho velikost a číslo PID paketu, který má přijmout. Pomocí systémového volání *ioctl* s parametrem DMX_SET_BUFFER_SIZE nastavím velikost bufferu (1x4096 = maximální velikost jedné sekce TS) pro vyfiltrovaná data. Následně vytvořím strukturu dmx_sct_filter_params, která slouží k nastavení filtru demuxu. Nastavím v ní požadovaný PID, jak dlouho ho má hledat (30000ms) a že má začít hledat okamžitě (DMX_IMMEDIATE_START), ověřit CRC součet přijatých paketů (DMX_CHECK_CRC) a po přijmutí první celé sekce složené s paketů s požadovaným PID ukončit činnost (DMX_ONESHOT). Filtr aplikuji opět pomocí volání *ioctl* s parametrem DMX_SET_FILTER. Na závěr použiji funkci *read*, která nahraje obsah demux bufferu do pole pro přijatá data. Jako návratová hodnota funkce *read* je délka přijatých dat, která je použita jako návratová hodnota fukce *readPid*.

```
#define TSBUFSIZE   (1 * 4096)
int readPid(int defd, __u8 * data, int dataSize, int pid) {
    long dmxBufferSize = TSBUFSIZE;

    if (ioctl(defd, DMX_SET_BUFFER_SIZE, dmxBufferSize) < 0) {
        perror("Set demux filter failed");
        return 0;
    }
    struct dmx_sct_filter_params sctFilterParams;
    memset(&sctFilterParams, 0, sizeof (struct dmx_sct_filter_params));
    sctFilterParams.pid = pid;
    sctFilterParams.timeout = 30000; //30s
    sctFilterParams.flags = DMX_IMMEDIATE_START | DMX_CHECK_CRC |
DMX_ONESHOT;

    if (ioctl(defd, DMX_SET_FILTER, &sctFilterParams) < 0) {
        perror("Set demux filter failed");
        return 0;
    }
```

```
    int len = read(defd, data, dataSize);
    if(len <= 0){
        perror("Read demux failed");
        return 0;
    }
    return len;
}
```

*Zdrojový kód 3: int readPid(int defd, __u8 * data, int dataSize, int pid)*

8.3.4 Funkce pro uložení XML souboru

Funkce *saveXMLDocument* (*Zdrojový kód 4*) přijímá jako vstupní parametr ukazatel na XML dokument, který chceme uložit. Ve funkci je nutné vyřešit možný paralelní přístup dvou aplikací k tomuto souboru. Na začátku nejprve pomocí funkce *open* získám deskriptor na požadovaný soubor v režimu čtení a zápisu (O_RDWR) , pokud soubor neexistuje bude vytvořen (O_CREAT). V případě vytvoření nového souboru je vytvořen s přístupovými právy 666. Pro to aby byly aplikovány přímo tyto práva je zapotřebí nastavit masku funkcí *umask* na 0. Pokud vše proběhne bez problému, pokusím se pomocí funkce *flock* získat exkluzivní zámek (LOCK_EX) na soubor, v případě, že je soubor již uzamčen funkce *flock* čeká až dojde k jeho uvolnění. Po získání zámku je uložena XML struktura do souboru a uvolněn zámek na souboru.

```
#define OUTPUTXMLDOCUMENTPATH "/tmp/dvbinfoout.xml"
int saveXMLDocument(pugi::xml_document* doc){
    int fd;
    umask(0);
    if((fd = open(OUTPUTXMLDOCUMENTPATH, O_RDWR|O_CREAT, 0666)) < 0){
        perror("Opening output file failed");
        return 0;
    }
    if(flock(fd, LOCK_EX) != 0){
        perror("Locking output file failed");
        return 0;
    }
    doc->save_file(OUTPUTXMLDOCUMENTPATH);
```

```
    flock(fd, LOCK_UN);
    close(fd);
    return 1;
}
```

Zdrojový kód 4: int saveXMLDocument(pugi::xml_document doc)*

8.3.5 Funkce upravující textové řetězce

Funkce *removeControlCharsFromBeginingOfString* (*Zdrojový kód 5*) odstraňuje kontrolní znaky vyskytující se na začátku textového řetězce. Jelikož se v několika případech vyskytovaly speciální znaky na začátku názvu poskytovatele TS nebo v názvech jednotlivých služeb, bylo zapotřebí tyto znaky odstranit, jelikož hlavně nula (0 = konec textového řetězce) způsobovala problémy při dalším zpracování. Funkce přijímá na vstupu pole znaků a jejich počet. Následně v cyklu prochází od začátku znak za znakem, až narazí na první ne-kontrolní znak zastaví se a vytvoří nové pole znaků již neobsahující tyto znaky na začátku. Funkce vrací odkaz na nově vytvořené pole znaků.

```
char* removeControlCharsFromBeginingOfString(unsigned char *text, int
len) {
    int newLen = 0;
    char *newText = new char[len+1];
    if (len <= 0) {
        newText[0] = 0;
        return newText;
    }
    int prohibitedCharsCount;
    for (prohibitedCharsCount = 0; prohibitedCharsCount < len;
prohibitedCharsCount++) {
        if (text[prohibitedCharsCount] >= 0 &&
text[prohibitedCharsCount] <= 32||text[prohibitedCharsCount] >= 127 &&
text[prohibitedCharsCount] <= 49824)
            continue;
        break;
    }
    newLen = len - prohibitedCharsCount;
    for (int j = 0; j < newLen; prohibitedCharsCount++, j++) {
```

```
        newText[j] = text[prohibitedCharsCount];
    }
    newText[newLen] = 0;
    return newText;
}
```

Zdrojový kód 5: char removeControlCharsFromBeginingOfString(unsigned char *text, int len)*

8.3.6 Ukázka dekodování PSI/SI tabulek

Prezentovaná funkce *processNITTable (Zdrojový kód 6)* ukazuje zpracování NIT tabulky. Z důvodů přehlednosti se jedná o velmi zjednodušenou verzi funkce použité v aplikaci DVBinfo, (vynechává zcela ukládání dat do XML souboru a dekóduje pouze některé deskriptory). K dekódování PSI/SI tabulek využívám knihovnu *libucsi*, která nabízí potřebné struktury k dekódování a řadu podpůrných funkcí.

Na počátku získám celou sekci NIT tabulky pomocí funkce *readPid*, tato sekce se dále rozkládá do struktur za pomocí funkcí knihovny libucsi. Velmi zajímavé je řešení opakovaného průchodu sekcí tabulky, kde se vyskytuje několik (0..N) deskriptorů. Používá se k tomu speciální zápis ve formátu *dvb_nit_section_descriptors_for_each(nit, curd) {…}*, kdy se jedná o makro, za kterým se skrývá klasický cyklus for.

Funkce *iprintf* je vyžívaná k formátovaní textového výstupu, kdy číslo jako první vstupní parametr znamená počet tabulátorů na začátku řádku.

Dekódování zbylých potřebných tabulek, vazby mezi nimi a ukládaní ukládaní dat do struktury XML souboru je možno nalézt ve zdrojovém kódu aplikace v příloze.

```
#define NIT_PID 16
#define TSSECTIONSIZE 4096
int processNITTable(int defd) {
    struct section *section;
    struct section_ext *section_ext = 0;
    __u8 data[TSSECTIONSIZE];

    int len = readPid(defd, data, sizeof (data), NIT_PID);
    if ((section = section_codec(data, len)) == 0) {
        fprintf(stderr, "NIT table: can't get section");
```

```
        return 0;
    }
    if (section->table_id == stag_dvb_network_information_actual) {
        struct dvb_nit_section *nit;
        struct descriptor *curd;
        struct dvb_nit_section_part2 *part2;
        struct dvb_nit_transport *cur_transport;

        if ((section_ext = section_ext_decode(section, 1)) == 0) {
            fprintf(stderr, "SCT XXXX NIT ext_section decode error\n");
            return 0;
        }
        printf("SCT Decode NIT (pid:0x%04x) (table:0x%02x)\n", NIT_PID,
section->table_id);
        if ((nit = dvb_nit_section_codec(section_ext)) == 0) {
            fprintf(stderr, "SCT XXXX NIT section decode error\n");
            return 0;
        }
        dvb_nit_section_descriptors_for_each(nit, curd) {
            if((curd->tag) == dtag_dvb_network_name){
                struct dvb_network_name_descriptor *dx;
                dx = dvb_network_name_descriptor_codec(curd);
                if (dx == NULL) {
                    fprintf(stderr, "DSC XXXX
dvb_network_name_descriptor decode error\n");
                    return 0;
                }
               iprintf(1, "DSC name: %s\n",
dvb_network_name_descriptor_name(dx),
dvb_network_name_descriptor_name_length(dx));
                }
```

```
        }
        part2 = dvb_nit_section_part2(nit);
        dvb_nit_section_transports_for_each(nit,part2, cur_transport) {
        printf("\tSCT transport_stream_id:0x%04x original_network_id:0x
%04x\n", cur_transport->transport_stream_id,
cur_transport->original_network_id);
        }
        dvb_nit_transport_descriptors_for_each(cur_transport, curd) {
              if (curd->tag == dtag_dvb_terrestial_delivery_system) {
                  struct dvb_terrestrial_delivery_descriptor *dx;
                  iprintf(2, "DSC Decode
dvb_terrestrial_delivery_descriptor\n");
                  dx =
dvb_terrestrial_delivery_descriptor_codec(curd);
                  if (dx == 0) {
                      fprintf(stderr, "DSC XXXX
dvb_terrestrial_delivery_descriptor decode error\n");
                      return 0;
                  }
                 iprintf(2, "DSC centre_frequency:%i bandwidth:%i
priority:%i time_slicing_indicator:%i mpe_fec_indicator:%i
constellation:%i hierarchy_information:%i code_rate_hp_stream:%i
code_rate_lp_stream:%i guard_interval:%i transmission_mode:%i
other_frequency_flag:%i\n", dx->centre_frequency, dx->bandwidth,
dx->priority, dx->time_slicing_indicator,dx->mpe_fec_indicator,
dx->constellation, dx->hierarchy_information,dx->code_rate_hp_stream,
dx->code_rate_lp_stream, dx->guard_interval, dx->transmission_mode,
dx->other_frequency_flag);
              }
          }
        }
    }
    return 1;
}
```

Zdrojový kód 6: int processNITTable(int defd)

Kompletní zdrojové soubory včetně Makefile souboru naleznete v přílohách ve složce DVBinfo. V příloze se také nachází ukázkový výstupní XML soubor a ukázkový výstup z aplikace. Při tvorbě aplikace DVBinfo jsem potřeboval informace o použití aplikací a knihoven *dvb-apps* a také dokumentaci k DVB API, obojí na stránkách jejich autorů [5]. Informace týkající se použití některých funkcí ze standardních knihoven jazyka C++ jsem nalezl převážně v [12]. Informace o použité XML knihovně Pugi na stránkách autorů [9]. Teoretické informace o konstrukci a struktuře PSI/SI tabulek jsem hledal v dokumentech [2] a [3], na stránkách věnujících se DVB [4] a [13] a pomocí aplikace *dvbsnoop* [11].

9 Moduly pro *snmpd* server

Dalším úkolem bylo získané informace o DVB vysílání pomocí aplikace DVBinfo poskytnout klientům prostřednictvím SNMP protokolu. Za tímto účelem jsem využil již existující SNMP démon pro OS Linux a rozšířil ho přídavnými moduly.

SNMP démon nabízí dvě možnosti rozšíření pomocí modulů, v prvním případě se moduly stanou součástí démonu a je nutné ho s přídavnými moduly kompilovat a tedy se připravit o využití oficiálního instalačního balíčku distribuovaného oficiálními kanály zvolené Linuxové distribuce. Proto jsem se rozhodl využít druhý, mírně složitější způsob, pro tvorbu modulu jako dynamicky načteného objektu. Kdy se pro jejich zprovoznění v konfiguračním souboru *snmpd* serveru pouze přidají řádky říkající, které moduly a odkud se mají načíst.

Implementovatelné informace z MIB souborů mají ve všech případech strukturu tabulky a tudíž moduly ve všech případech využívají konstrukce pro implementaci MIB tabulky.

Všechny, krom jednoho modulu fungují jako prostřednicí mezi XML souborem s daty a SNMP serverem a jeden modul si získává potřebná data přímo sám a nabízí téměř okamžitou hodnotu. Tedy v tomto směru existují principiálně dva odlišné přístupy k tvorbě modulu.

9.1 Použité technologie

- Moduly jsou napsány v programovacím jazyce C.
- K tvorbě základní struktury modulu jsem využil aplikaci *mib2c*.
- Pro práci s XML využívám C knihovnu libxml2.
- Moduly vyvíjím ve vývojovém prostředí Netbeans.

9.2 Společné náležitosti modulů

Každý dynamicky načítaný modul musí implementovat minimálně dvě funkce. Funkci pro inicializaci a deinicializaci.

Pro inicializaci slouží funkce *void init_module_name (void)* (*Zdrojový kód 7*). Na počátku vytvořím strukturu pro ukládaní dat. Jelikož se jedná o implementaci tabulky, proto volím strukturu typu *netsnmp_table_data_set*.

Pomocí funkce *netsnmp_table_set_add_indexes* nastavím index budoucí tabulky, indexem bude první sloupec tabulky typu integer. Pokud potřebujeme více indexů, jednoduše do funkce dáme další datové typy oddělené čárkami. Pak musíme v další části ve funkci *netsnmp_table_set_multi_add_default_row* definující hlavičku tabulky dodržet správné přiřazení pořadí sloupců.

Funkcí *netsnmp_create_handler_registration* vytvořím strukturu obsahující informace potřebné pro registraci handleru poskytující data pro odpověď, při požadavku týkající se implementované části MIB souboru, kterou specifikuji uvedenou hodnotou OID. V tomto případě je handlerem právě datová struktura typu *netsnmp_table_data_set* se jménem table_set, kterou jsem předchozími kroky vytvořil. Použitím funkce *netsnmp_register_table_data_set* zaregistruji *table_set*, tím řeknu SNMP serveru, aby v případě požadavku na data z tabulky, pro kterou je napsán tento modul, použil k odpovědi data z vytvořené datové struktury data_set.

Nyní ještě musím zajistit, aby se *table_set* naplňoval aktuálními daty. Tuto problematiku jsem řešil nejprve přidáním odkazu na funkci při vytváření struktury *netsnmp_handler_registration*, kdy místo *NULL* byl vložen zaobalený odkaz na funkci, která se pak volala pokaždé při požadavku na data z registrovaného *table_set*. Toto řešení sice bylo relativně funkční, ale velmi neefektivní z důvodů častého přístupu k datům. Proto jsme pátral jak tento problém nejlépe vyřešit, až jsem narazil na *cache_handler.*

Cache handler vytvořím funkcí *netsnmp_get_cache_handler*, kde nastavím čas platnosti cache, odkaz na funkci pro vytvoření nové cache, odkaz na funkci pro odstranění staré cache a OID, pro specifikaci k jaké položce z MIB souboru se vztahuje. Následně ho ještě zaregistruji do řetězce handlerů funkcí *netsnmp_inject_handler*, kde použiji stejnou strukturu *netsnmp_create_handler_registration* jako pro registraci *data_set*. Tímto se *cache_handler* bude volat vždy před *data_set* a v případě, že už uplyne nastavený čas, zavolá se funkce pro načtení cache, ve které inicializuji *table_set* a až po inicializaci teprve odešle SNMP server data z *table_set*. U statických dat jsem zvolil dobu platnosti dat na 60 sekund a u dynamických na 5 sekund. Tímto krokem sice u dynamických dat mírně snížím aktuálnost, ale snížím také počty přístupu k DVB zařízení při častých požadavcích. Funkce pro odstranění již neplatné cache se zavolá automaticky po uplynutí její platnosti.

```
#define COLUMN_MGTSINPUTNUMBER              1
#define COLUMN_MGTSID                       2
#define COLUMN_MGTSORIGINALNETWORKID        3
#define COLUMN_MGTSNETWORKID                4
#define COLUMN_MGTSNETWORKNAME              5
netsnmp_table_data_set *table_set;
static oid mgTSTable_oid[] = {1,3,6,1,4,1,2696,3,3,1,1,2};

void init_mgTSTable(void)
{
    DEBUGMSGTL(("mgTSTable","Initializing the mgTSId module\n"));
```

```
    DEBUGMSGTL(("mgTSTable","Preparing registration\n"));
     table_set = netsnmp_create_table_data_set("mgTSTable");
     netsnmp_mib_handler *cache_handler;
     /*Adding indexes*/
    DEBUGMSGTL(("mgTSTable", "adding indexes to table mgTSTable\n"));
    netsnmp_table_set_add_indexes(table_set,
                    ASN_INTEGER, /* index: mgTSInputNumber */
                                        0);
    DEBUGMSGTL(("mgTSTable", "adding column types to table
mgTSTable\n"));
    netsnmp_table_set_multi_add_default_row(table_set,
                  COLUMN_MGTSINPUTNUMBER, ASN_INTEGER, 0, NULL, 0,
                  COLUMN_MGTSID, ASN_INTEGER, 0, NULL, 0,
                  COLUMN_MGTSORIGINALNETWORKID, ASN_INTEGER, 0, NULL, 0,
                  COLUMN_MGTSNETWORKID, ASN_INTEGER, 0, NULL, 0,
                  COLUMN_MGTSNETWORKNAME, ASN_OCTET_STR, 0, NULL, 0, 0);
      /* registering the table with the master agent */
    netsnmp_handler_registration *reginfo =
        netsnmp_create_handler_registration("mgTSTable", NULL,
        mgTSTable_oid, OID_LENGTH(mgTSTable_oid), HANDLER_CAN_RWRITE);


    netsnmp_register_table_data_set(reginfo, table_set, NULL);


    DEBUGMSGTL(("mgTSTable","Done initalizing mgTSTable module\n"));


    cache_handler = netsnmp_get_cache_handler(
        LONGCACHETIMEOUT, /* how long a cache is valid for */
        cache_load,  /* a pointer to the cache loading function */
        cache_free,  /* a pointer to the cache freeing function */
        mgTSTable_oid,
        OID_LENGTH(mgTSTable_oid));   /* the OID of the registration
```

```
point */
    netsnmp_inject_handler(reginfo, cache_handler);
}
```

Zdrojový kód 7: void init_mgTSTable(void)

Pro deinicializaci slouží funkce *void deinit_module_name (void)* (*Zdrojový kód 8*). Funkce odstraní *table_set a* odregistruje všechny handlery registrované na základě OID.

```
void deinit_mgTSTable(void)
{
    netsnmp_delete_table_data_set(table_set);
    table_set = NULL;
    unregister_mib(mgTSTable_oid,OID_LENGTH(mgTSTable_oid));
}
```

Zdrojový kód 8: void deinit_mgTSTable(void)

9.3 Modul načítající data z XML souboru

Takto jsem implementoval všechny vybrané tabulky z DVB-MGSIGNALCHARACTERISTICS-MIB. Jako ukázku jsem zvolil modul implementují mgTSTable.

Funkce *cache_load* (*Zdrojový kód 9*) zajistí načtení dat do *table_set* z XML souboru. Data získávám z XML pomocí xpath dotazu. Pro načtení XML dokumentu jsem vytvořil funkci *getXMLdoc*, pokud se dokument podaří úspěšně načíst, program pokračuje dál ve zpracování, jestliže dojde k problému, aplikace ponechá předchozí obsah *table_set*. Při úspěšném načtení dokumentu odstraním všechny staré záznamy z *table_set* a zavolám funkci *getNodesPtr*, která nad dokumentem aplikuje xpath dotaz k vyselektováni uzlů obsahující data pro naplnění konkrétní MIB tabulky.

V následném cyklu postupně zpracovávám všechny vybrané uzly a nahrávám hodnoty do předem připravených proměnných. V případě že nějaký záznam chybí, je v proměnné uložená hodnota -1. Získaná data vložím do struktury reprezentující jeden řádek v *table_set*, který posléze přidám do table_set. Po skončení cyklu procházejícího vybrané uzly z XML souboru volám funkci *closeXML*, aby dealokovala prostředky potřebné pro zpracování XML. Tímto je funkce *cache_load* u konce.

Princip zbývajících modulů je obdobný, liší se pouze obsahem cyklu zpracovávajícího uzly vybrané z XML souboru, jelikož každá MIB tabulka obsahuje rozdílné záznamy.

```
#define INPUTXMLDOCUMENTPATH "/tmp/dvbinfoout.xml"
int cache_load (netsnmp_cache *cache, void *magic)
```

```
{
    DEBUGMSGTL(("mgTSTable","Load Handler\n"));
    xmlDocPtr doc;
    xmlChar *xpath = (xmlChar*)
"/dvb/mg/mgSignalCharacteristics/mgSignalCharacteristicsObjects/mgTSStr
ucture/mgTSTable/mgTSEntry";
    xmlNodeSetPtr nodeset;
    xmlXPathObjectPtr result;
    xmlNodePtr cur;
    int i;
    netsnmp_table_row *row;
    int mgTSInputNumber = -1;
    int mgTSId = -1;
    int mgTSOriginalNetworkID = -1;
    int mgTSNetworkID = -1;
    xmlChar * mgTSNetworkName;
    doc = getXMLDoc();
    if(doc == NULL){
        DEBUGMSGTL(("mgTSTable","Problem with loading file %s, keeping
old values in table.\n", INPUTXMLDOCUMENTPATH));
        return SNMP_ERR_NOERROR;
 }
 cleanTableSet(table_set);
 result = getNodesPtr(doc, xpath);

   if (result) {
        nodeset = result->nodesetval;
        for (i = 0; i < nodeset->nodeNr; i++) {
            cur = nodeset->nodeTab[i]->xmlChildrenNode;
            while (cur != NULL) {
                if ((!xmlStrcmp(cur->name, (const xmlChar
```

```
*)"mgTSInputNumber"))){
                    mgTSInputNumber = atoi (xmlNodeListGetString(doc,
cur->xmlChildrenNode, 1));
                    DEBUGMSGTL(("mgTSTable","mgTSInputNumber:
%d\n",mgTSInputNumber));
      }
                else if ((!xmlStrcmp(cur->name, (const xmlChar
*)"mgTSId"))){
                    mgTSId = atoi (xmlNodeListGetString(doc, cur-
>xmlChildrenNode, 1));
                    DEBUGMSGTL(("mgTSTable"," mgTSId: %d\n",mgTSId));
      }
                else if ((!xmlStrcmp(cur->name, (const xmlChar
*)"mgTSOriginalNetworkID"))){
                    mgTSOriginalNetworkID = atoi
(xmlNodeListGetString(doc, cur->xmlChildrenNode, 1));
                    DEBUGMSGTL(("mgTSTable","mgTSOriginalNetworkID:
%d\n",mgTSOriginalNetworkID));
      }
                else if ((!xmlStrcmp(cur->name, (const xmlChar
*)"mgTSNetworkID"))){
                    mgTSNetworkID = atoi (xmlNodeListGetString(doc,
cur->xmlChildrenNode, 1));
                    DEBUGMSGTL(("mgTSTable","mgTSNetworkID:
%d\n",mgTSNetworkID));
      }
                else if ((!xmlStrcmp(cur->name, (const xmlChar
*)"mgTSNetworkName"))){
                    mgTSNetworkName = xmlNodeListGetString(doc, cur-
>xmlChildrenNode, 1);
                    DEBUGMSGTL(("mgTSTable","mgTSNetworkName:
%s\n",mgTSNetworkName));
      }
            cur = cur->next;
```

```
    }
        row = netsnmp_create_table_data_row();
        netsnmp_table_row_add_index(row, ASN_INTEGER, &mgTSInputNumber,
sizeof(mgTSInputNumber) );
        netsnmp_set_row_column(row, 2 ,ASN_INTEGER, &mgTSId,
sizeof(mgTSId) );
        netsnmp_set_row_column(row, 3 ,ASN_INTEGER,
&mgTSOriginalNetworkID, sizeof(mgTSOriginalNetworkID) );
        netsnmp_set_row_column(row, 4 ,ASN_INTEGER, &mgTSNetworkID,
sizeof(mgTSNetworkID) );
        netsnmp_set_row_column(row, 5 ,ASN_OCTET_STR, mgTSNetworkName,
strlen(mgTSNetworkName) );
        netsnmp_table_dataset_add_row(table_set, row);
        mgTSInputNumber = -1;
        mgTSId = -1;
        mgTSOriginalNetworkID = -1;
        mgTSNetworkID = -1;
        mgTSNetworkName = 0;
        }
  }else{
        DEBUGMSGTL(("mgTSTable","Data loaded from file %s is empty.\n",
INPUTXMLDOCUMENTPATH));
  }
        closeXML(doc, result);
        return SNMP_ERR_NOERROR;
  }
```

*Zdrojový kód 9: int cache_load (netsnmp_cache *cache, void *magic)*

Funkce *cache_free (Zdrojový kód 10)* je volána automaticky po vypršení času platnosti cache. V mém řešení, kdy zachovávám v případě problémů s načtením XML dokumentu původní data, tuto funkci nevyužívám a data odstraňuji až po úspěšném připojení k XML.

```
void cache_free (netsnmp_cache *cache, void *magic)
{
```

```
    DEBUGMSGTL(("mgTSTable","Free Handler\n"));
}
```

*Zdrojový kód 10: void cache_free (netsnmp_cache *cache, void *magic)*

9.3.1 Funkce pro načtení XML dokumentu

V případě načítání XML dokumentu *(Zdrojový kód 11)* nastává opět problém s možným paralelním přístupem k souboru. Na počátku získám funkcí *open* souborový deskriptor na XML dokument, určený pouze ke čtení (O_RDONLY). Následně řeším problém paralelního přístupu funkcí *flock*, která se snaží získat sdílený zámek (LOCK_SH) na XML soubor a také využívá neblokující režim (LOCK_NB). Neblokujícím režimem zamezím velké prodlevě a možnému zdržení odpovědi serveru, kdy může dojít až k vypršení časového limitu pro odpověď. Po načtení dokumentu uvolním zámek a jako návratovou hodnotu použiji načtený dokument.

```
xmlDocPtr getXMLDoc (void) {
    int fd = open(INPUTXMLDOCUMENTPATH, O_RDONLY);
    if(flock(fd, LOCK_SH | LOCK_NB) != 0){
        perror("Cant get lock");
        return NULL;
  }
    xmlDocPtr doc;
    doc = xmlParseFile(INPUTXMLDOCUMENTPATH);
    flock(fd, LOCK_UN);
    close(fd);
    if (doc == NULL ) {
            fprintf(stderr,"Document not parsed successfully. \n");
            return NULL;
  }
    return doc;
}
```

Zdrojový kód 11: xmlDocPtr getXMLDoc (void)

9.4 Modul načítající dynamická data

Tento typ modulu je použit pro implementaci tabulky mgDHCTable z MIB souboru DVB-

FRONTENDCHARACTERISTICS-MIB. Tento modul získává informace z frontend části DVB přijímače. Tyto informace jsou dostupné prakticky okamžitě po připojení na frontend zařízení, takže je není potřeba předem zaznamenávat.

Po prvotní deklaraci potřebných proměnných *(Zdrojový kód 12)* následuje hlavní cyklus, který prochází postupně všechny možně dostupná frontend zařízení. Toho docílím za využití textového řetězce "/dev/dvb/adapter%d/frontend%d" a funkce *snprintf*, která doplňuje čísla adaptérů a frontendu. Pokud pokus o připojení k zařízení dopadne úspěšně, začnou se na zařízení posílat dotazy za použití systémového volání *ioctl*.

U aktuální přijímané frekvence signálu nemůžeme jednoduše použít získanou hodnotu, jelikož význam získaného čísla je závislý na typu použité modulace přijímaného signálu a na jejím základě se musí výsledná aktuální frekvence vypočítat.

Po každém úspěšném cyklu jsou získaná data vložena do struktury reprezentující řádek tabulky a ten je následně přidán do *table_set*. Průchodem a ověřením všech frontend zařízení funkce končí.

```
#define FRONTENDDEVICE "/dev/dvb/adapter%d/frontend%d"
#define MAXDEVICECOUNT 8
int cache_load(netsnmp_cache *cache, void *magic) {
    DEBUGMSGTL(("mgDCHTable","load from mgDCHTable \n"));
    int mgDCHInputNumber = 1;
    int mgDCHSignalStrength = -1;
    int mgDCHSignalBER = -1;
    int mgDCHSignalSNR = -1;
    int mgDCHSignalUncorrectedBlocks = -1;
    int mgDCHSignalFrequency = -1;

    unsigned int adapter  = 0;
    unsigned int frontend = 0;
    char fedev[128];
    int fefd, z;
    __u16 signal_strlength;
    __u16 snr;
    __u32 ber;
    __u32 uncorrected_block;
```

```
    struct dvb_frontend_info fe_info;
    struct dvb_frontend_parameters fe_param;
    netsnmp_table_row *row;
    while(mgDCHInputNumber <= MAXDEVICECOUNT){
        snprintf(fedev, sizeof(fedev), FRONTENDDEVICE, adapter,
frontend);

        if ((fefd = open(fedev, O_RDONLY | O_NONBLOCK)) < 0) {
            mgDCHInputNumber++;
            adapter++;
            perror("Open Frontend error");
            DEBUGMSGTL(("mgDCHTable","cannot open frontend \n"));
            continue;
        }
        if( ioctl(fefd, FE_READ_SIGNAL_STRENGTH, &signal_strlength) >=
0){
            mgDCHSignalStrength = (signal_strlength)*100/(0xffff);
        }
        else{
            mgDCHSignalStrength = 0;
        }
        if( ioctl(fefd, FE_READ_SNR, &snr) >= 0){
          mgDCHSignalSNR = (snr)*100/(0xffff);
        }
         if( ioctl(fefd, FE_READ_BER, &ber) >= 0){
          mgDCHSignalBER = ber;
        }
         if( ioctl(fefd, FE_READ_UNCORRECTED_BLOCKS, &uncorrected_block)
>= 0){
          mgDCHSignalUncorrectedBlocks = uncorrected_block;
        }
```

```
        if( ioctl(fefd, FE_GET_INFO, &fe_info) >= 0){
            if( ioctl(fefd, FE_GET_FRONTEND, &fe_param) >= 0){
                if ( fe_info.type == FE_OFDM ){
                    if(fe_param.frequency+10600000 >= 11700000) {
                        mgDCHSignalFrequency =
(fe_param.frequency+10600000)/1000;
                    }
                    else {
                        mgDCHSignalFrequency =
(fe_param.frequency+9750000)*1000; }
                }
                else if ( fe_info.type == FE_QAM || fe_info.type ==
FE_QPSK ){
                    if(fe_param.frequency+10600000 >= 11700000) {
                        mgDCHSignalFrequency =
(fe_param.frequency+10600000);
                    }
                    else { mgDCHSignalFrequency =
(fe_param.frequency+9750000)*1000000;}}
                else { mgDCHSignalFrequency = -1;}
            }
            else { mgDCHSignalFrequency = -1; }
        }
        else { mgDCHSignalFrequency = -1; }
        close(fefd);
        row = netsnmp_create_table_data_row();
        netsnmp_table_row_add_index(row, ASN_INTEGER,
&mgDCHInputNumber, sizeof(mgDCHInputNumber) );
        netsnmp_set_row_column(row, 2 ,ASN_INTEGER,
&mgDCHSignalStrength, sizeof(mgDCHSignalStrength) );
        netsnmp_set_row_column(row, 3 ,ASN_INTEGER, &mgDCHSignalBER,
sizeof(mgDCHSignalBER) );
```

```
        netsnmp_set_row_column(row, 4 ,ASN_INTEGER, &mgDCHSignalSNR,
sizeof(mgDCHSignalSNR) );
        netsnmp_set_row_column(row, 5 ,ASN_INTEGER,
&mgDCHSignalUncorrectedBlocks, sizeof(mgDCHSignalUncorrectedBlocks) );
        netsnmp_set_row_column(row, 6 ,ASN_INTEGER,
&mgDCHSignalFrequency, sizeof(mgDCHSignalFrequency) );
        netsnmp_table_dataset_add_row(table_set, row);
        mgDCHInputNumber++;
        adapter++;
        mgDCHInputNumber = 1;
        mgDCHSignalStrength = -1;
        mgDCHSignalBER = -1;
        mgDCHSignalSNR = -1;
        mgDCHSignalUncorrectedBlocks = -1;
        mgDCHSignalFrequency = -1;
    }
    return SNMP_ERR_NOERROR;
}
```

*Zdrojový kód 12: int cache_load(netsnmp_cache *cache, void *magic)*

Funkce *cache_free* (*Zdrojový kód 13*) po vypršení platnosti cache vymaže obsah *table_set*.

```
void cache_free(netsnmp_cache *cache, void *magic) {
    DEBUGMSGTL(("mgDCHTable", "Free Handler\n"));
    cleanTableSet(table_set);
}
```

*Zdrojový kód 13: void cache_free(netsnmp_cache *cache, void *magic)*

Kompletní zdrojové soubory všech modulů včetně souboru Makefile naleznete v přílohách. Pro vytvoření modulů jsem čerpal převážně z [7] , informace o používání zvolené XML knihovny jsem nalezl v [10]. Informace o použitých funkcích z knihoven jazyka C jsem převážně nalezl v [12].

10 Instalační balíček

Pro zajištění uživatelské přívětivosti jsem vytvořil standardizovaný instalační balíček dostupný pro OS založené na debianu. Balíček je dostupný jak v 32-bitové tak i v 64-bitové verzi.

Instalátor zajistí:

- Nainstaluje aplikaci DVBinfo.
- Nastaví automatické hodinové spouštění DVBinfo jako cron úlohu.
- Pokud není již nainstalován, nainstaluje SNMP server (*snmpd*).
- Nahraje MIB moduly pro *snmpd* server.
- Přidá záznam o nově přidaných modulech do nastavení *snmpd*.

10.1 Použité nástroje

- Nástroj *dh_make* pro vytvoření struktury deb balíčku.
- Nástroj *lintian* pro ověření kvality balíčku.
- Nástroj *debconf* pro zajištění interakce s uživatelem během instalace.
- Virtual Box pro testování balíčku ve virtuálním prostředí.

10.2 Tvorba balíčku

Pro vytvoření balíčku jsem sestavil adresářovou strukturu, hlavní složka dvbsnmp_1.0 obsahuje tři podsložky dvbinfo, snmpdplugins a MIBs. Každá ze tří podsložek obsahuje vlastní Makefile a hlavní složka obsahuje hlavní Makefile.

Použitím programu dh_make jsem vytvořil v hlavním adresáři čtvrtý adresář s názvem debian, obsahující výchozí a ukázkové informace o budoucí podobě a funkci instalačního balíčku, které jsem dále editoval dle svých požadavků.

Součástí složky debian jsou také soubory:

control – Základní konfigurační soubor balíčku (Výpis 6). Obsahuje základní informace o balíčku (název, autor...) a také seznam závislostí na ostatních balíčcích. Pro instalaci mého balíčku musí být nainstalován *snmpd* server a aplikace *debconf* pro zobrazení otázek během instalace. Pro jeho kompilaci musí být dostupné balíčky *debhelper*, *dvb-apps*, *libsnmp-dev* a *libxml2-dev*.

Source: dvbsnmp

```
Section: misc
Priority: optional
Maintainer: Lukas Zajac <zajac.ov@gmail.com>
Build-Depends: debhelper (>= 9), dvb-apps (>=1.1.1), libsnmp-dev (>=5.7.1), libxml2-dev (>=2.9.1)
Standards-Version: 3.9.4

Package: dvbsnmp
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}, snmpd (>=5.7.1), debconf (>= 0.2.26)
Description: Monitor your DVB server using SNMP.
 DVBSNMP implements parts of DVB-MGSIGNALCHARACTERISTICS-MIB, so you can easily monitor
 state of your DVB server using SNMP protocol. DVBSNMP extends Net-SNMP daemon (snmpd) by
 dynamic loadable plugins.
```

Výpis 6: Soubor control

dirs – Textový soubor (Výpis 7) obsahující cesty ke složkám, kam se při provedení příkazu *make install* mají nahrát soubory. Tyto složky se vytvoří uvnitř deb balíčku a během instalace balíčku se jejich obsah překopíruje do skutečné adresářové struktury, to znamená že se před uvedené cesty dá lomítko reprezentující kořenový adresář. Do adresáře /usr/bin se během instalace nahraje aplikace *dvbinfo*, v adresáři /usr/bin/snmpd jsou naistalovány MIB moduly a v adresáři /usr/share/snmp/mibs budou nahrány implementované MIB soubory.

```
usr/bin
usr/lib/snmpd
usr/share/snmp/mibs
```

Výpis 7: Soubor dirs

config – Skript spuštěný během instalace (Výpis 8), který získá díky interakci s uživatelem odpovědi na dvě otázky. Zda se má spustit aplikace *dvbinfo* a zda se má restartovat *snmpd* server pro načtení nové konfigurace. Pro interakci používám nástroj *debconf*. Na začátku přidám podporu pro funkce debconf. Funkce *db_input* připraví otázku k zobrazení. Parametr *high* zajistí nejvyšší prioritu zobrazení uživateli a *dvbsnmp/run_dvbinfo* specifikuje, která šablona se má pro otázku použít. *Db_go* zobrazí otázku.

```
#!/bin/sh -e
. /usr/share/debconf/confmodule
```

```
db_input high dvbsnmp/run_dvbinfo || true
db_go
db_input high dvbsnmp/restart_snmpd || true
db_go
```

Výpis 8: Config skript

templates – Soubor obsahuje definice dat (Výpis 9), které zobrazuji uživateli pomocí *debconf*. *Obsahuje* dvě otázky s odpovědí ano/ne a dvě textové informace pro uživatele.

```
Template: dvbsnmp/run_dvbinfo
Type: boolean
Description: Do you want run BVBinfo now?
 Do you want run BVBinfo now, or you can run it manually later or wait for automatic run in cron.

Template: dvbsnmp/restart_snmpd
Type: boolean
Description: Restart snmpd server now?
 For apply changes SNMP sevrer needs to be restarted. You can restart in now or manually later.

Template: dvbsnmp/restore_exist
Type: note
Description: snmpd.comf exist.
 "It was created backup of config file /etc/snmpd/snmpd.conf. If you have any problems with SNMP server after changes in config file, simply remove new file and restore backup file."

Template: dvbsnmp/restore_no_exist
Type: note
Description: snmpd.comf not exist.
 "You are probably using different config file for your SNMP server. It was created config file /etc/snmpd/snmpd.conf. If you have any problems with SNMP server after changes in config file, simply remove newly created file."
```

Výpis 9: Template soubor

postinst – Skript spuštěný po dokončení instalace (Výpis 10). Ve skriptu zajišťuji konfiguraci hlavně

snmpd serveru. Na začátku připojuji podporu *debconf*. Pokud existuje konfigurační soubor pro *snmpd* je vytvořena jeho záloha a uživatel je o této skutečnosti informován prostřednictvím *debconf*. V případě neexistence souboru je na tuto skutečnost uživatel také upozorněn. Následně je do existujícího nebo vytvořeného konfiguračního souboru přidány řádky s obsahem *dlmod název_modulu cesta_k_modulu*, což zajistí jejich nahrání při dalším zapnutí *snmpd* serveru.

Příkazem *usermod* přidám uživatele snmp (tento uživatel spouští v init skriptu *snmpd* server) do skupiny video. To je potřebné z důvodů abych zajistil přístup k DVB zařízením pro modul s přímým přístupem k datům.

Následně příkazem *db_get* získám odpověď uživatele na položenou otázky v config skriptu. Kde je podle parametru *dvbsnmp/run_dvbinfo* určena otázka. Pokud je odpověď na spuštění *dvbinfo* kladná, je aplikace spuštěna. Jestliže je kladná odpověď na druhou otázku o restartování *snmpd* serveru, dojde k jeho restartování.

Jedná se pouze o část postinst skriptu, celý skript naleznete v příloze.

```
. /usr/share/debconf/confmodule
FILE="/etc/snmp/snmpd.conf"
if [ -f $FILE ]; then
    cp $FILE /etc/snmp/snmpd.conf.bak
    db_input high dvbsnmp/restore_exist || true
    db_go
else
    db_input high dvbsnmp/restore_no_exist || true
    db_go
fi
echo "# dvbsnmp -> load dynamic plugins:" >> $FILE
echo "dlmod mgTSTable /usr/lib/snmpd/dvbmodules/mgTSTable.so" >> $FILE
echo "dlmod mgDCHTable /usr/lib/snmpd/dvbmodules/mgDCHTable.so" >> $FILE
echo "dlmod mgServiceTable /usr/lib/snmpd/dvbmodules/mgServiceTable.so" >> $FILE
echo "dlmod mgPIDTable /usr/lib/snmpd/dvbmodules/mgPIDTable.so" >> $FILE
echo "dlmod mgNITDeliverySystemTable /usr/lib/snmpd/dvbmodules/mgNITDeliverySystemTable.so" >>
$FILE
echo "dlmod mgServiceECMTable /usr/lib/snmpd/dvbmodules/mgServiceECMTable.so" >> $FILE
echo "dlmod mgEMMTable /usr/lib/snmpd/dvbmodules/mgEMMTable.so" >> $FILE
```

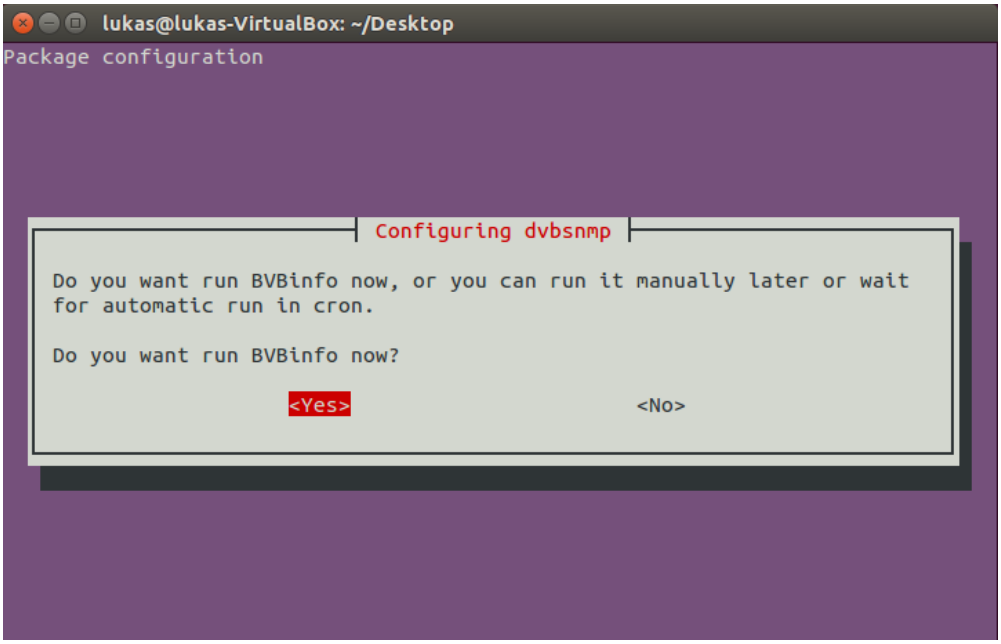
```
echo "dlmod mgPIDECMTable /usr/lib/snmpd/dvbmodules/mgPIDECMTable.so" >> $FILE
usermod -a -G video snmp
db_get dvbsnmp/run_dvbinfo
if [ "$RET" = true ]; then
        dvbinfo || true
fi
db_get dvbsnmp/restart_snmpd
if [ "$RET" = true ]; then
        service snmpd restart
fi
```

Výpis 10: Postinst skript

cron.hourly – Tento skript je během instalace nahrán do složky /etc/cron.hourly se jménem dvbsnmp a jak původní název napovídá, skript se bude spouštět automaticky v hodinovém intervalu. Tímto způsoben řeším automatické spouštění aplikace *dvbinfo*, aniž by musela běžet jako démon na pozadí.

10.3 Instalace

Pro instalaci si nejprve musím zjistit zda server používá 32-bitový, nebo 64-bitový OS na základě čehož zvolím správnou verzi instalačního balíčku.



Obrázek 8: Průběh instalace

Příkazem

```
sudo dpkg -i dvbsnmp_1.0-1_platforma.deb
```

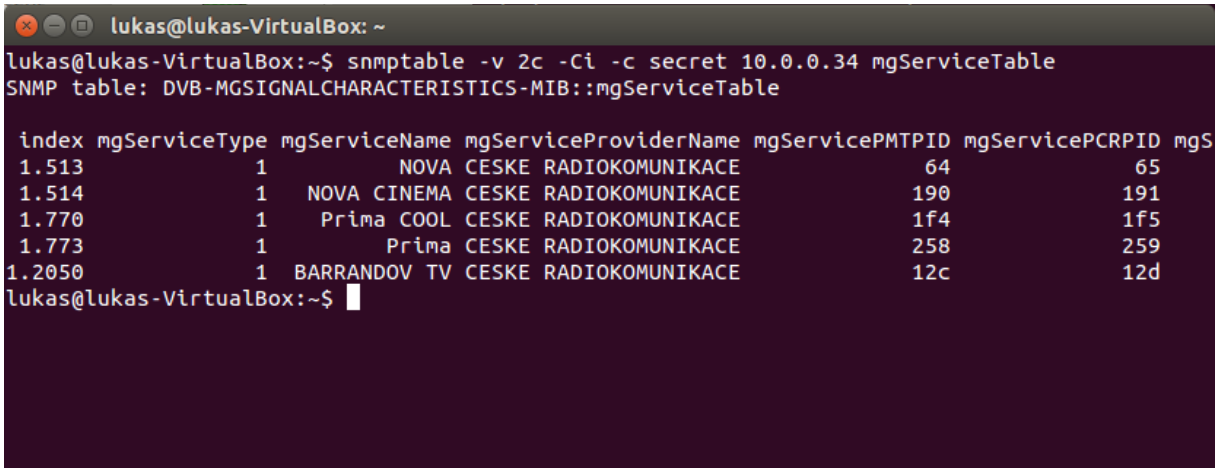
v případě chybějících závislostí použiji příkaz pro jejich doinstalaci.

```
sudo apt-get install -f
```

Během instalace uživatel zodpoví položené otázky potřebné k dokončení instalace (Obrázek 8) .

Pokud jsem před instalací neměl nakonfigurován SNMP server, je nutné v případě požadavku přístupu mimo localhost upravit jeho konfiguraci. Bližší informace o konfiguraci *snmpd* jsem nalezl například v [7].

Ověření funkčnosti SNMP a DVB modulů můžeme vidět na obrázku 9.



Obrázek 9: Výstup snmptable

Kompletní zdrojovou složku obsahující vše potřebné k vytvoření deb balíčku naleznete v přílohách ve složce Instalátor, kde také naleznete již hotové instalační balíčky. Pro tvorbu instalačního balíčku jsem čerpal převážně z [14] a o použití nástroje *debconf* jsem nalezl informace z [15] a manuálových stránek programu.

11 Cacti

Poslední kapitola popisuje tvorbu modulu pro Cacti. Aplikace Cacti slouží k prezentaci dat ve formě grafů. Nejčastěji se používá pro monitoring síťového provozu a umožňuje přijímat data prostřednictvím SNMP protokolu.

Po analýze dat, které jsem o DVB schopen získat a poskytovat přes SNMP protokol, připadají v úvahu pouze data získaná z frontend zřízení. Tyto data se nacházejí v tabulce mgDCHTable.

Studiem možností nabízených Cacti jsem se rozhodl pro vytvoření takzvaného datového dotazu. Ve své podstatě se jedná o připojení Cacti na celou MIB tabulku. Aby toto bylo možné provést, je zapotřebí vytvořit XML dokument popisující strukturu této tabulky, pomocí které bude Cacti získávat data z SNMP serveru.

XML dokument (Výpis 11) mapující mgDCHTable z DVB-FRONTENDCHARACTERISTICS-MIB. V tagu *query* se nachází popis celého data query. Podstatnou položkou je *oid_index*, který určuje oid indexu v tabulce.

V přístupu k indexu můžou nastat dva problémy, v některých případech je v MIB tabulce vícerozměrný index, který zde nelze použít. Dalším větším problémem je, že podle definice by měly být položky indexu v tabulce nepřístupné (jakožto i v mém případě), čímž je nemohu získat. Řešením obou problému je použití *oid_index_parse*. V *oid_index* uvedu OID na první dostupnou položku v tabulce a Cacti si pomocí funkce snmpwalk projde celý sloupec, získaná OID na jednotlivé položky ve sloupci se poté zpracují regulárním výrazem uvedeným v *oid_index_parse* a všechny čísla oddělené tečkami za posledním číslem v *oid_index* znamenají čísla indexu a vytvoří index pro Cacti. Podle velikosti indexu musím upravit regulární výraz, v aktuální podobě přijímá poslední číslo, jelikož mgDCHTbale má jednorozměrný index.

Tagy *index_order* a *index_order_type* určují řazení při zobrazování uživateli. Řádky jsou řazeny podle mgDCHInputNumber a podle velikosti jeho hodnoty. V tagu *fields* jsou obsaženy jednotlivé sloupce v tabulce. Tag *source* pro jednotlivé položky určuje zdroj dat, pokud se jedná o hodnou získanou parsováním OID použiji jako zdroj index, pro data získávána přímo přes SNMP použiji value. Tagem *direction* určím, zda se jedná o položku pouze informativní (input), nebo se bude zobrazovat v grafu (output).

```
<query>
    <name>Get SNMP DVB mgDCHTable</name>
    <description>Queries a host for a mgDCHTable</description>
    <oid_index>.1.3.6.1.4.1.2696.3.10.1.1.1.2</oid_index>
    <oid_index_parse>OID/REGEXP:.*\.([0-9]{1,2})$</oid_index_parse>
```

```
    <index_order>mgDCHInputNumber</index_order>
    <index_order_type>numeric</index_order_type>
    <fields>
        <mgDCHInputNumber>
            <name>Input Number</name>
            <method>walk</method>
            <source>index</source>
            <direction>input</direction>
        </mgDCHInputNumber>
        <mgDCHSignalStrenth>
            <name>Signal Stregth</name>
            <method>walk</method>
            <source>value</source>
            <direction>output</direction>
            <oid>.1.3.6.1.4.1.2696.3.10.1.1.1.2</oid>
        </mgDCHSignalStrenth>
    </fields>
</query>
```

Výpis 11: Datový dotaz

V uvedeném XML dokumentu nejsou uvedeny všechny položky, kompletní XML soubor naleznete v adresáři Cacti pod názvem mgDCHTable.xml v přílohách.

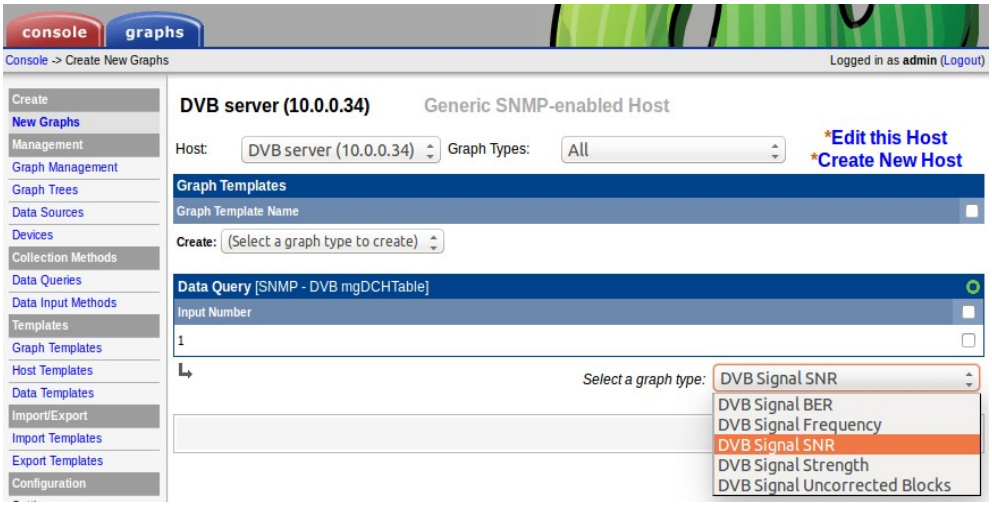
Po úspěšném načtení vytvořeného datového dotazu jsem vytvořil šablonu pro datový zdroj a šablony pro grafy (síla signálu, BER, SNR, frekvence, chybovost) využívající šablony datového zdroje. Tyto nastavení jsem vyexportoval do XML souboru, který slouží k přenosu a snadnému znovunahrání do nových instalací Cacti.

11.1 Instalace a použití modulu

Instalaci modulu do Cacti provedu volbou importu šablony, z adresáře Cacti v přílohách vyberu soubor cacti_data_query_snmp_-_dvb_mgdchtable.xml. Po přidání tohoto modulu je zapotřebí abych spojil tento datový dotaz s nějakým zřízením (reprezentuje server, kde je specifikována jeho IP adresa a nastavení SNMP), čímž bude datový dotaz vědět s jakými parametry má volat SNMP požadavky.

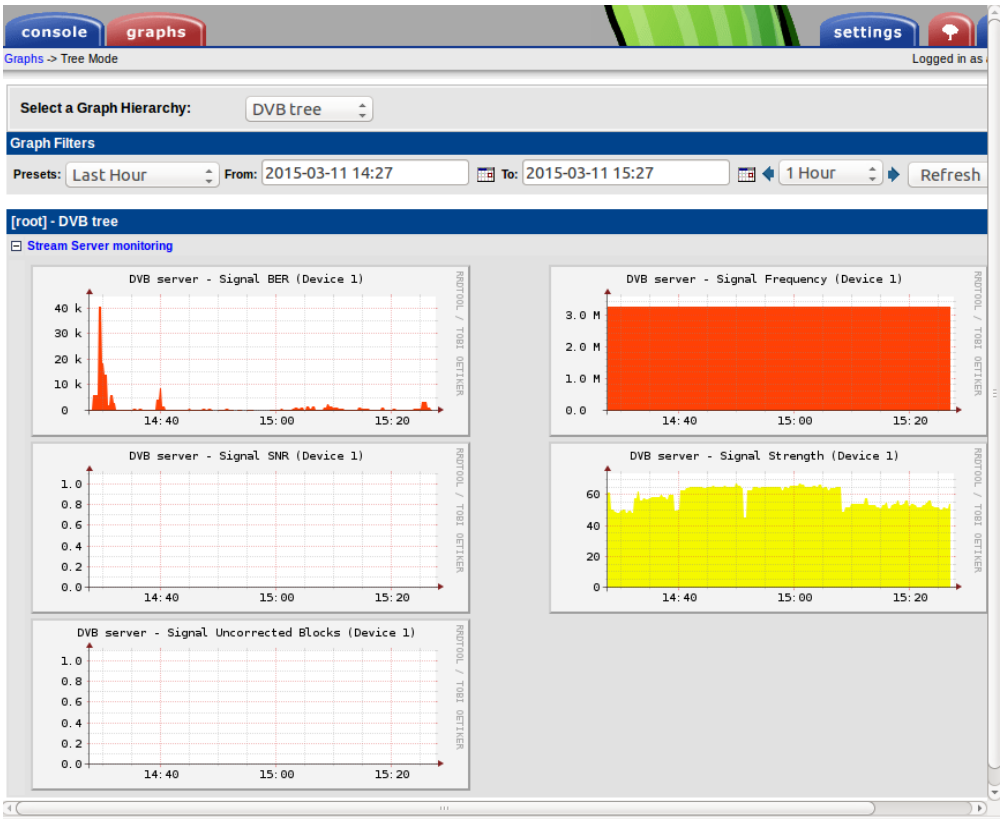
Nyní už pouze zvolím přidat nový graf, vyberu požadované zařízení a z nabídky datových dotazů

vyberu řádek z tabulky podle čísla adaptéru (mgDCHInputNumber neboli Input Number) a typ grafu který chci vytvořit. Princip přidání nového grafu využívajících služeb modulu je vidět na obrázku 10.



Obrázek 10: Vytvoření nového grafu

Obrázek 11 ukazuje všechny dostupné grafy, umožňuje modul vytvořit pro jedno zařízení.



Obrázek 11: Grafy

Cacti

Informace pro tvorbu modulu a práci s Cacti jsem čerpal na stránkách věnovaných tomuto programu [6]. V přílohách ve složce Cacti najdete všechny zmiňované zdrojové soubory.

12 Závěr

Cílem práce bylo vytvořit komplexní nástroj pro monitorování DVB vysílacího serveru. Toto obnášelo analýzu problému a nalezení nejvhodnějšího postupu řešení a následnou implementaci. Implementoval jsem nástroj pro získávaní dat z PSI/SI tabulek TS a dynamicky načítané moduly pro *snmpd* server. Ze všech vytvořených součástí jsem následně vytvořil instalační baliček, který zajistí vše potřebné pro snadné nasazení na DVB vysílací server. Nakonec jsem za účelem prezentace dynamických dat vytvořil modul pro monitorovací nástroj Cacti.

Pro úspěšné dokončení práce jsem musel detailně nastudovat vlastnosti a strukturu MPEG TS a zvláště pak PSI/SI tabulky. Dalším pro mě zajímavým problémem bylo vytvoření dynamický načítaných modulů pro *snmpd* server, který byl složitější než jsem původně očekával. Toto bylo způsobeno faktem, že jsem nikde nenalezl potřebné informace pro tvorbu pokročilejších modulů a vycházel jsem pouze z velmi základních příkladů.

Celá vytvořená sada nástrojů je plně funkční a připravena k plnému nasazení. Do budoucna by bylo možno vytvořit jednotnou aplikaci za účelem prezentace statických dat i dynamických dat, která by nahradila Cacti a nástroje z balíku aplikací *snmp*. Tímto krokem by se zajisté zlepšila přehlednost a ještě více zvýšila uživatelská přívětivost.

Veškeré zdrojové soubory a instalační balíčky jsou veřejně dostupné na GitHubu [8].

Seznam použité literatury

[1]MAURO, Douglas R a Kevin J SCHMIDT. *Essential SNMP*. 2nd ed. Beijing: O´Reilly, 2005, 442 s. ISBN 05-960-0840-6.

[2]ETSI. *Digital Video Broadcasting (DVB): Specification for Service Information (SI) in DVB systems* [online]. 2010 [cit. 2015-02-11]. Dostupné z:http://www.etsi.org/deliver/etsi_en/300400_300499/300468/01.11.01_60/en_300468v011101p.pdf

[3]ETSI. *Digital Video Broadcasting (DVB): SNMP MIB for test and measurement applications in DVB systems* [online]. 2002 [cit. 2015-02-09]. Dostupné z: http://www.etsi.org/deliver/etsi_ts/102000_102099/102032/01.01.01_60/ts_102032v010101p.pdf

[4]DVB. *Digital Video Broadcasting (DVB): Guidelines on implementation and usage of Service Information (SI)* [online]. 2013 [cit. 2015-02-16]. Dostupné z: https://www.dvb.org/resources/public/standards/a005_dvb-si_impl_guide.pdf

[5]*Linux TV: Television with Linux* [online]. [cit. 2015-02-11]. Dostupné z: http://www.linuxtv.org/

[6]*Cacti: Complete network graphing solution* [online]. [cit. 2015-03-11]. Dostupné z: http://www.cacti.net/

[7]NET-SNMP. *Net-SNMP* [online]. [cit. 2015-02-11]. Dostupné z: http://www.net-snmp.org/

[8]ZAJAC, Lukáš. *Dvbsnmp: Monitor your DVB server using SNMP* [online]. [cit. 2015-04-10]. Dostupné z: https://github.com/MrLukass/dvbsnmp.git

[9]*Pugi XML: The C++ XML Parser* [online]. [cit. 2015-02-11]. Dostupné z: http://pugixml.org/

[10]*The XML C parser and toolkit of Gnome* [online]. [cit. 2015-02-11]. Dostupné z: http://xmlsoft.org/

[11]*Dvbsnoop: DVB / MPEG stream analyzer* [online]. [cit. 2015-02-11]. Dostupné z: http://dvbsnoop.sourceforge.net/

[12]*The C++ Resources Network* [online]. [cit. 2015-02-11]. Dostupné z: http://www.cplusplus.com/

[13]Program-specific information. [online]. [cit. 2015-02-11]. Dostupné z: http://en.wikipedia.org/wiki/Program-specific_information

[14]HowToPackageForDebian. [online]. [cit. 2015-03-05]. Dostupné z: http://wiki.debian.org/HowToPackageForDebian

[15]HESS, Joey. *The Debconf Programmer's Tutorial* [online]. 1999 [cit. 2015-03-05]. Dostupné z: http://www.fifi.org/doc/debconf-doc/tutorial.html

Přílohy

I. Příloha na CD

Obsah CD:

- Zdrojové kódy aplikace DVBinfo.
- Ukázkové výstupy z aplikace DVBinfo.
- Zdrojové kódy modulů pro SNMP server.
- Soubory pro vytvoření instalačního balíčku.
- Instalační balíček ve 32-bitové i 64-bitové verzi.
- Modul pro Cacti.
- MIB soubory.
- Použité zdroje dostupné v pdf.